



**This electronic thesis or dissertation has been
downloaded from Explore Bristol Research,
<http://research-information.bristol.ac.uk>**

Author:
Bewley, Tom

Title:
Tree Models for Interpretable Agents

General rights

Access to the thesis is subject to the Creative Commons Attribution - NonCommercial-No Derivatives 4.0 International Public License. A copy of this may be found at <https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode>. This license sets out your rights and the restrictions that apply to your access to the thesis so it is important you read this before proceeding.

Take down policy

Some pages of this thesis may have been removed for copyright restrictions prior to having it been deposited in Explore Bristol Research. However, if you have discovered material within the thesis that you consider to be unlawful e.g. breaches of copyright (either yours or that of a third party) or any other law, including but not limited to those relating to patent, trademark, confidentiality, data protection, obscenity, defamation, libel, then please contact collections-metadata@bristol.ac.uk and include the following information in your message:

- Your contact details
- Bibliographic details for the item, including a URL
- An outline nature of the complaint

Your claim will be investigated and, where appropriate, the item in question will be removed from public view as soon as possible.



**This electronic thesis or dissertation has been
downloaded from Explore Bristol Research,
<http://research-information.bristol.ac.uk>**

Author:
Bewley, Tom

Title:
Tree Models for Interpretable Agents

General rights

Access to the thesis is subject to the Creative Commons Attribution - NonCommercial-No Derivatives 4.0 International Public License. A copy of this may be found at <https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode>. This license sets out your rights and the restrictions that apply to your access to the thesis so it is important you read this before proceeding.

Take down policy

Some pages of this thesis may have been removed for copyright restrictions prior to having it been deposited in Explore Bristol Research. However, if you have discovered material within the thesis that you consider to be unlawful e.g. breaches of copyright (either yours or that of a third party) or any other law, including but not limited to those relating to patent, trademark, confidentiality, data protection, obscenity, defamation, libel, then please contact collections-metadata@bristol.ac.uk and include the following information in your message:

- Your contact details
- Bibliographic details for the item, including a URL
- An outline nature of the complaint

Your claim will be investigated and, where appropriate, the item in question will be removed from public view as soon as possible.

Tree Models for Interpretable Agents

By

TOM BEWLEY



Department of Engineering Mathematics
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of DOCTOR OF PHILOSOPHY in the Faculty of Engineering.

SEPTEMBER 2023

Word count: sixty six thousand and twenty five

Abstract

As progress in AI impacts all sectors of society, the world is destined to see increasingly complex and numerous autonomous decision-making agents, which act upon their environments and learn over time. These agents have many practical applications, but also pose risks that demand ongoing human oversight. The field of AI interpretability builds tools that help human stakeholders understand the structure and origins of agent behaviour. This aids various downstream tasks, including verifying whether that behaviour is safe and reliable. In this thesis, we take several new perspectives on the interpretability problem, developing models that give insight into how agents respond to the state of their environments and vice versa, how they learn and change over time, and how this process is guided by the objectives supplied by humans themselves. While shifting between these perspectives, we maintain a coherence of approach by basing all of our methods on a theory of abstraction with rule-based models called trees.

We begin this thesis by formalising the tree abstraction framework and investigating its foundations. We then instantiate concrete examples of tree models for representing the behaviour of an agent not just in terms of its state-to-action policy, but also via its value function and state dynamics. We exploit the rule-based tree structures to generate textual explanations of agent actions over time and propose novel visualisations of behavioural trends across the environment state space. From this point, we further develop the concept of trees as dynamics models, using a contrastive objective uncover the changes that occur during agent learning. We visualise these models with graphs and heatmaps, and show how prototype trajectories can be identified to summarise an agent’s behaviour at each stage of learning.

In the second half of the thesis, we shift perspective to use trees as reward functions for training agents themselves, which provides an interpretable grounding for learnt behaviour. Furthermore, we show how reward trees can be learnt from human feedback, thereby drawing a connection between interpretability and the literature on human-agent alignment. We establish the efficacy of reward tree learning via experiments with synthetic and real human feedback on four benchmark tasks, then explore its interpretability benefits, showing how it enables detailed monitoring of an agent’s learning progress. After further refining the reward tree learning method, we evaluate it in the industrially-motivated use case of aircraft handling and develop a new interpretability technique that is synergistic with model-based agent architectures.

By blending quantitative and qualitative evaluations across a range of environments, we aim to show how our methods are broadly applicable and provide complementary insights that could be combined in a unified interpretability toolkit. We view both our individual tree models, and our diverse but cohesive research strategy, as meaningful contributions to the interpretability community. However, much work remains to be done, including completing thorough user evaluations in real-world domains, developing more scalable and optimal tree learning algorithms, and extending our methods to systems of multiple interacting agents.

Author's Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..



DATE: 21ST SEPTEMBER 2023

Acknowledgements and Dedication

The effort of forcing oneself through the motivational treacle of PhD research is at once brutally individual and necessarily collective. Although I am far from the researcher I hope to be one day, my progress so far has largely been due to the input of others, and for that, I will always be grateful. As I look back, finally, on the past four years, I have many heartfelt thanks to give.

Firstly, to my supervisors, Prof. Jonathan Lawry and Prof. Arthur Richards, for their complementary expertise and unwavering support through all the doubts and tangents. To Ben, Rachel, Ian and Freddy at Thales for offering fresh perspectives, real-world grounding, and a listening ear. To all the staff and fellow students at the Alan Turing Institute for the most enjoyable, productive and collaborative year of my PhD and for helping me fall deeper in love with London. To Paul Harper and the entire Engineering Design family for fostering the most inspirational undergraduate experience I could have hoped for, which gave me the confidence to embark on something so different and scary afterwards. To the communities and organisations behind the software stack upon which my research code is built (PyTorch, Gymnasium, Weights & Biases...), as well as the excellent Bristol HPC team. And to `splendid-microwave-108`, a particularly plucky soft actor-critic RL agent, who provided 750 rip-roaring behavioural episodes for my analytical delectation.

A deeper thanks still goes to my kind, ambitious and downright ridiculous friends, whose protracted bad example reminds me that life is really about capsizing canoes, post-COVID music, and full-commitment fancy dress. To my parents and brother, who have given unfaltering love in the face of my objectively silly decision to pursue a postgraduate research degree, and to whom I feel more connected and indebted with every passing year. And above all, to Tash, my profoundly special wife, who has been here by my side for every ebb and flow of this existential log flume. Tash, you make my life better every day. You make me better every day. You make the world better every day. I have no idea where we're going, but I know the journey will be great because we are making it together. So let's keep making it.

This thesis is dedicated to my grandparents: Doreen and Charlie Bewley, Pamela and Donald Lloyd-Jones. If life is an MDP, they may well have solved it. It turns out the optimal policy was simple all along...

Table of Contents

	Page
List of Figures	xi
List of Key Notation	xv
1 Introduction	1
1.1 The Need for Interpretable Agents	1
1.2 Agents and their Environments	3
1.3 The Agent Interpretability Landscape	5
1.3.1 Defining Interpretability	5
1.3.2 Challenges of the Agent Context	6
1.3.3 Intrinsic Methods	8
1.3.4 Mechanistic Methods	9
1.3.5 Behaviourist Methods	10
1.4 Themes, Gaps and Opportunities	12
1.5 Thesis Contributions	14
1.6 Research Output	16
2 Abstraction with Trees	17
2.1 Introduction	17
2.2 Symbols and Abstractions	17
2.3 From Conceptual Spaces to Tree Abstractions	18
2.3.1 Convexity	19
2.3.2 Partitioning	19
2.3.3 Hierarchy	20
2.3.4 Axis-alignment	21
2.4 Query-Efficient Abstractions	23
2.5 Abstractions of Agents and Environments	25
2.6 Conclusion	28

3	Tree Models of Agent Behaviour	31
3.1	Introduction	31
3.2	Tree Models of Black Box Policies	32
3.2.1	Generic Problem Statement	32
3.2.2	Recursive Feature Generation	33
3.2.3	Tree-Structured Policy Model	34
3.2.4	Modelling Procedure	36
3.3	Traffic Simulator Experiments	40
3.3.1	Traffic Simulator Environment	40
3.3.2	Target Policies	41
3.3.3	Recursive Generation of the Maximal Feature Space	41
3.3.4	Dataset and Training	42
3.3.5	Baselines	44
3.4	Quantitative Evaluation	44
3.4.1	Evaluation by Predictive Accuracy	44
3.4.2	Evaluation by On-policy Divergence	45
3.4.3	Evaluation by Mean Time Between Failures	47
3.4.4	Correlations between Quality Metrics	48
3.5	Model Interpretation and Explanation	49
3.5.1	Tree Diagram Comparison	49
3.5.2	Factual Local Explanation	51
3.5.3	Counterfactual Local Explanation	52
3.5.4	Temporal Explanatory Stories	55
3.6	Pit Stop: Reviewing the Policy Modelling Approach	57
3.6.1	Related Work	57
3.6.2	What is Missing in a Policy Model?	59
3.7	TRIPLETREE: A Multiattribute Tree Abstraction	59
3.7.1	Data Preparation and Model Structure	60
3.7.2	Learning Algorithm	63
3.7.3	Related Work	64
3.8	Quantitative Evaluation	65
3.9	Model Interpretation and Explanation	67
3.9.1	Multiattribute Visualisation in Feature Space	68
3.9.2	Factual, Counterfactual and Temporal Explanation	69
3.9.3	Hypothetical Trajectories	72
3.10	Experiments in a Higher-dimensional Environment	74
3.10.1	Hyperrectangle Projection for $D > 2$	75
3.10.2	Hyperrectangle Slicing and Visual Counterfactuals	78

TABLE OF CONTENTS

3.10.3 Hypothetical Trajectories	79
3.11 Conclusion	80
4 Tree Models of Agent Learning	83
4.1 Introduction	83
4.1.1 Related Work	85
4.2 Theory of Contrastive Spatiotemporal Abstraction	85
4.2.1 The Contrastive Objective	87
4.2.2 Temporal Abstraction	89
4.3 Tree Abstraction Algorithms	91
4.3.1 Contrastive State Abstraction (CSA)	91
4.3.2 Contrastive Spatiotemporal Abstraction (CSTA)	92
4.3.3 Pseudocode and Subfunction Details	94
4.4 Scaling of Jensen-Shannon Divergence with m and n	96
4.4.1 Theoretical Analysis for m	96
4.4.2 Theoretical Analysis for n	98
4.4.3 Empirical Validation	99
4.5 2D Maze Experiment	100
4.5.1 State Abstraction Process	101
4.5.2 Temporal Abstraction Process	102
4.5.3 Pairwise Window Divergence	103
4.5.4 Visitation and Transition Time Series	104
4.5.5 Transition Graph Comparison	105
4.6 LunarLander Experiment	106
4.6.1 Abstraction Structure	106
4.6.2 Visitation Time Series	107
4.6.3 Transition Graph Comparison	107
4.6.4 Window Prototypes	109
4.6.5 Posterior analysis and Counterfactual Review	109
4.7 Comparison of Abstraction Algorithm Variants	111
4.8 Conclusion	114
5 Tree Models of Human Preferences	117
5.1 Introduction	117
5.1.1 Motivation	118
5.1.2 Related Work	119
5.2 Preference-based Reward Learning	120
5.3 Interpretable Reward Learning with Trees	122
5.3.1 Trajectory-Level Return Estimation	123

5.3.2	Leaf-level Reward Estimation	124
5.3.3	Tree Growth	126
5.3.4	Tree Pruning	127
5.4	Offline and Online Learning Algorithms	129
5.4.1	Trajectory Provenance and Diversity	131
5.4.2	Growth Initiation, Stopping and Resumption	131
5.4.3	Optimistic Active Sampling	132
5.4.4	Scheduling of Online Trajectory and Preference Collection	133
5.5	Experimental Setup	134
5.5.1	Environments and Tasks	134
5.5.2	Common Parameters	136
5.6	Quantitative Performance: Oracle Preferences	137
5.6.1	Offline Setting	137
5.6.2	Online Setting	139
5.7	Quantitative Performance: Human Preferences	141
5.7.1	Offline Setting	141
5.7.2	Online Setting	144
5.8	Summary of Key Findings	146
5.9	Interpretability Analysis	147
5.9.1	Failure Case: RoboCar using Offline Human Preferences	147
5.9.2	Success Case: 2D Maze using Online Oracle Preferences	149
5.10	Conclusion	152
6	A Use Case for Reward Trees	155
6.1	Introduction	155
6.2	Aircraft Handling Use Case	156
6.2.1	Motivation	156
6.2.2	Implementation	158
6.3	Methodological Improvements	159
6.3.1	Trajectory-Level Return Estimation	160
6.3.2	Tree Growth and Pruning	162
6.3.3	Model-based RL Agents	163
6.3.4	Online Learning Setup	164
6.4	Experiments and Results	166
6.4.1	Common Parameters	166
6.4.2	Online Performance Evaluation	167
6.4.3	Policy-Invariant Evaluation	169
6.4.4	Visual Trajectory Inspection	171
6.4.5	Sensitivity Analysis	173

TABLE OF CONTENTS

6.4.6	Comparison to Model-free Reinforcement Learning	174
6.5	Interpretability Analysis	176
6.5.1	Tree Structure Appraisal	176
6.5.2	Leaf-level Alignment	177
6.5.3	Trajectory Report Card	178
6.5.4	Preference-based Reward Explanation	179
6.6	Explaining Model-based Action Selection	180
6.7	Conclusion	183
7	Conclusions and Further Work	185
7.1	Review of Contributions	185
7.2	Practical Uses of Proposed Models	185
7.3	Successes of the Overall Approach	188
7.4	Limitations of the Overall Approach	188
7.5	Further Work	189
	Bibliography	191

List of Figures

Figure	Page
1.1 The agent-environment interaction cycle and generic reinforcement learning process.	3
1.2 The three perspectives on agent interpretability.	7
1.3 The parable of <i>The Blind Men and the Elephant</i>	15
2.1 A conceptual space in $(\mathbb{R}^D, \mathbb{Y})$ form, and four geometric desiderata for abstractions.	20
2.2 Query-centric abstraction efficiency in $(\mathbb{R}^D, \mathbb{Y})$ form conceptual spaces.	24
2.3 A conceptual spaces model of agent-environment interaction.	27
3.1 Generic problem setup for interpretable policy modelling. The objective is to minimise the loss between a_t and $a'_t \forall t \in \{1, \dots, N\}$ while ensuring both the feature function ϕ and policy model π' are comprehensible under human scrutiny.	33
3.2 A state-action dataset as observations in a conceptual space in $(\phi_{\text{all}}(\mathcal{S}), \mathbb{Y})$ form, where $\phi_{\text{all}}(\mathcal{S})$ is the maximal state feature space and the only attribute is the agent's action, $\mathbb{Y} = \{\mathcal{A}\}$. Here, \mathcal{A} contains five discrete actions, $\mathcal{A} = (--, -, 0, +, ++)$, which match those in our traffic simulator experiments. Also shown are tree abstractions at three stages during the modelling procedure: splitting the root, growing to zero impurity, and pruning.	37
3.3 Left: A selection of track topologies for the traffic environment, with vehicles shown as coloured rectangles and junctions represented by red circles. Right: Definition of the six features used by π_F , as measured by the blue vehicle.	40
3.4 Curves of validation set accuracy and number of features used across the pruning sequences for each target policy. The rightmost point in each curve corresponds to the unpruned tree \mathcal{X} . Numbered vertical lines indicate the trees chosen to carry forward to evaluation.	43
3.5 Left: Track topologies. Right: Model accuracy on an unseen test set $\mathcal{D}_{\text{test}}$	46
3.6 Expected divergence from target policy after 1000 timesteps.	46
3.7 Mean time between failures across 100 episodes of up to 1000 timesteps.	46
3.8 Correlations between model rankings on all topologies and quality metrics.	48

3.9	Ground truth tree and learnt tree model (pruning level 5) for the fully-learnable policy π_F . Leaf labels indicate their modal actions as follows: $---\rightarrow -\frac{v_{\max}}{4}$; $-\rightarrow -\frac{v_{\max}}{12}$; $0\rightarrow 0$; $+\rightarrow \frac{v_{\max}}{12}$; $++\rightarrow \frac{v_{\max}}{4}$. Split thresholds used in rules are rounded to 2 d.p.	50
3.10	To find the minimal counterfactual for state s^* and foil action a' , we start by identifying the set of leaves satisfying the foil condition $X_{(\text{action}=a')}$ ($= \{x_1, \dots, x_5\}$ here), then find the closest point to $\mathbf{f}^* = \phi(s^*)$ on the boundary of each. We filter first by the 0-norm of the corresponding δ vectors; this removes $\mathbf{f}^{(*\rightarrow x_1)}$ and $\mathbf{f}^{(*\rightarrow x_5)}$ from contention. We then sort by 2-norm (faint green circle), which identifies $\mathbf{f}^{(*\rightarrow x_4)}$ as the minimal counterfactual.	53
3.11	Conceptual space representation of TRIPLETREE.	61
3.12	The 2-dimensional road environment.	65
3.13	Prediction losses for four variants of the road environment, with R_{left} , R_{right} , R_{speed} as stated in the left-hand labels.	66
3.14	Analysis of worst-case loss across the space of weight vectors for trees of various sizes. Coloured dots show the identity of the worst loss (blue: action, red: value, green: derivatives) at the 21 weightings tested. Heatmaps show the magnitude of the worst loss as a ratio of the loss from a one-leaf tree, linearly interpolated between test locations.	67
3.15	Five types of visualisation using TRIPLETREE.	68
3.16	Various kinds of rule-based explanation for action and value.	70
3.17	Illustrating the inadequacy of previous methods for counterfactual explanation in the temporal context, and the proposed MBB-based approach.	71
3.18	Hypothetical trajectories in the road environment, and example of a trajectory overlaid onto action and value visualisations.	74
3.19	Training and validation losses in LunarLander.	75
3.20	Hyperrectangle projection process for $D = 3$. Colours represent the leaf-level summary statistic to be visualised, such as mean action or value, or observation density. Notice that the final colour of each rectangular area is an average of the leaf cuboids above it.	76
3.21	Partial dependence plots for the LunarLander policy.	77
3.22	Displaying visual counterfactuals for the LunarLander policy on ICE plots.	78
3.23	Hypothetical trajectories for the LunarLander policy.	80
4.1	Application of state abstraction to a $k = 3$ -policy transition dataset.	87
4.2	Two state abstraction options and their resultant JSD values.	89
4.3	Two temporal abstraction options and their resultant JSD values.	90

4.4	Scaling of JSD with m and n for random trajectories in $[0, 1]^2$ with various velocity and noise parameters v, σ , and when selecting both random axis-aligned splits (orange) and those that maximise the contrastive abstraction objective (blue). Top row: scaling with m . Bottom row: scaling with n for the specific values of m shown.	100
4.5	2D Maze environment, learning curve and transition data for 750 learning episodes.	101
4.6	Visualisation of state abstraction process for 2D Maze environment.	102
4.7	Visualisation of temporal abstraction process for 2D Maze environment.	103
4.8	Additional results figures for 2D Maze environment. Note: \emptyset = episode termination.	104
4.9	LunarLander, learning curve and transition data for 500 random/learning episodes.	106
4.10	Abstraction results and analysis for LunarLander environment.	108
4.11	Posterior analysis of an unsuccessful (crash) landing during the random phase. . .	110
4.12	Posterior analysis of a successful landing late in learning.	110
4.13	Distribution of regularised JSD rankings of 24 algorithm variants across 25 regularisation settings for each of three datasets of learning agent transitions. Variants sorted by median rank across all datasets and regularisation settings (ties broken by mean rank). Box plot whiskers located at $1.5 \times \text{IQR}$ below/above lower/upper quartiles respectively.	113
4.14	Comparison of original and ‘best’ variants across all 75 evaluation cases. First four columns show normalised differences (i.e. $(\text{best} - \text{original})/\text{best}$) in the stated metrics, with > 0 indicating that ‘best’ improves (regularised) JSD and < 0 being preferable for m and n . Final column compares runtimes of the two variants; < 1 indicates that ‘best’ is faster.	113
5.2	Pairwise trajectory preferences as a directed graph.	121
5.3	The four stages of reward tree learning applied to a toy example of $K = 4$ preferences over $N = 4$ trajectories in a $D = 2$ -feature space (Figure 5.2 duplicated above for reference).	125
5.4	Flow diagram of online reward tree learning.	131
5.5	Empirical pair sampling probabilities (averaged over 100 repeats) for $K_{\max} = 2000$ preferences over $N_{\max} = 100$ trajectories, which arrive in batches of size $N_{\text{batch}} = 10$. The preference batch size scheduling (Equation 5.15) and the recency constraint (Equation 5.16) must be implemented in tandem to correct the earliness bias fully and achieve uniformity.	133
5.6	The four RL environments used in experiments.	134
5.7	Performance in offline setting using oracle preferences; additional plots for 2D Maze.	138
5.8	Performance in online setting using oracle preferences; learning timeline for RoboCar.	140
5.9	Performance in offline setting using human preferences; additional plots for Pendulum.	142
5.10	Performance in online setting using human preferences; results with learning timelines for 2D Maze and LunarLander.	145

LIST OF FIGURES

5.11	Tree diagram and other visualisations for a failure case in RoboCar.	149
5.12	Tree diagram and other visualisations for a success case in 2D Maze.	151
6.1	State-action space of aircraft handling domain, and diagrams of all three tasks. . .	158
6.2	Comparison of old (least squares matrix) and new (NLL gradient) methods of trajectory-level return estimation for a toy example of $K_{\max} = 1000$ noise-free oracle preferences over $N_{\max} = 100$ trajectories with normally distributed ground truth returns. 20 repeats completed; all results plotted as scatter points.	161
6.3	Time series of metrics for online NN- and tree-based reward learning.	169
6.4	Policy-invariant evaluation of online NN- and tree-based reward learning.	170
6.5	Agent trajectories using the best models by ORR (oracle and random for comparison).	172
6.6	Comparative sensitivity analysis of reward learning with NNs and trees.	174
6.7	Comparing the use of PETS and SAC agents on the Follow task.	175
6.8	Diagram of a reward tree learnt for the Chase task (“r” denotes reward).	177
6.9	Alignment of leaf-level reward predictions with ground truth oracle rewards.	178
6.10	Report card explaining the rewards predicted for a trajectory.	178
6.11	Explaining differences in leaf reward predictions by preference reversal.	179
6.12	Explaining PETS agent actions with doubly-decomposed reward differences.	182

List of Key Notation

$s_t \in \mathcal{S}$:	The state of an environment at time t , in the state space \mathcal{S} .
$a_t \in \mathcal{A}$:	The action of an agent at time t , in the action space \mathcal{A} .
$\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$:	The agent's policy function, which specifies a distribution over actions a_t to take in each state s_t .
$T : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$:	The environment's dynamics function, which specifies a distribution over next states s_{t+1} given the current state s_t and agent action a_t .
$R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$:	A reward function, which maps (state, action, next state) transitions to real-valued rewards.
$\phi : \text{dom} \rightarrow \mathbb{R}^D$:	A feature function, which maps a domain (usually the state space \mathcal{S}) to a D -dimensional real feature space.
$d \in \{1, \dots, D\}$:	The index of one feature in a feature space.
$\mathbf{f} \in \mathbb{R}^D$:	A feature vector, where \mathbf{f}_d denotes values of the d th feature.
$x \subseteq \mathbb{R}^D$:	A subset of a feature space, containing a number of feature vectors.
$\mathcal{X} = \{x_1, x_2, \dots, x_m\}$:	An abstraction of a feature space into m subsets (<i>Note: in a tree abstraction, the properties of convexity, partitioning, hierarchy and axis-alignment hold and the subsets are called leaves</i>).
\mathcal{D}	:	A dataset of observations of an agent behaving in its environment (<i>Note: the contents of each element varies between chapters</i>).
$\mathcal{D}_x \subseteq \mathcal{D}$:	Given a tree abstraction \mathcal{X} , the subset of observations in \mathcal{D} whose corresponding feature vectors belong to leaf $x \in \mathcal{X}$.
$I(\mathcal{D}_x) \in \mathbb{R}_{\geq 0}$:	A non-negative impurity measure for the observations in leaf $x \in \mathcal{X}$.
$c \in \mathcal{C}_d$:	A candidate threshold at which to split a leaf along the d th feature, in the finite set $\mathcal{C}_d \subset \mathbb{R}$.
$x^{(d \geq c)}$ and $x^{(d < c)}$:	The new leaves that would be created by splitting leaf x at threshold c along feature d .
$Q(x, d, c) \in \mathbb{R}$:	The quality of the split defined by x , d and c , which may be defined in terms of the impurities $I(\mathcal{D}_x)$, $I(\mathcal{D}_{x^{(d \geq c)}})$ and $I(\mathcal{D}_{x^{(d < c)}})$.

Chapter 1

Introduction

1.1 The Need for Interpretable Agents

In the broadest popular framing, the field of artificial intelligence (AI) aims to construct rational agents, which pursue objectives by acting upon their environments [218]. In recent years, AI agents have been built that beat champion game players [231], drive autonomous vehicles [11], manage complex building control problems [257], predict and affect economic systems [258], and tackle core challenges in sustainable energy [64]. The present rate of progress is unprecedented in the history of the field, and shows little sign of slowing. However, a consistent lesson, received bitterly by some [239], is that this success has been mostly driven not by novel insight into the algorithms underlying intelligent behaviour, but by equipping agents with generic statistical models with many tuneable parameters, and leveraging massive datasets and computation to optimise the parameters for a given objective. This approach is referred to as machine learning.

While the effectiveness of eschewing explicit human priors in favour of *tabula rasa* learning is remarkable and humbling, it risks sacrificing something once thought core to AI as a scientific discipline. As agents and their learning algorithms grow more complex and unconstrained by human intuition, our ability to understand their internal representations, and explain the origins of their visible behaviour, is becoming increasingly strained.

One might be tempted to remain undisturbed by this trend as long as impressive performance breakthroughs continue to occur, but such an attitude would be problematic. This is because AI agents do not exist in a vacuum, but are rather embedded in a modern society in which technology has a profound impact on people's lives. As such, there is a wide range of human stakeholders with an interest in maintaining oversight over how agents behave and learn. For example, consider the following *user stories* [50]:

- **As an owner of an AI agent, I want to** verify its capabilities and limitations in as-yet-unseen scenarios, **so that** I can deploy it with confidence.
- **As a user of an AI agent, I want to** understand how my actions are influencing its behaviour, **so that** I gain some control over the behaviour produced in future.

- **As a** regulator of technology, **I want to** screen proposed AI agents for pernicious functionality and biases, **so that** I can protect the public from harm.
- **As an** AI practitioner, **I want to** trace the effects of changes I make to an agent model and its learning algorithm, **so that** I can make improvements efficiently.
- **As an** AI researcher, **I want to** develop scientific insight into the mechanisms and emergent properties of agent learning, **so that** this can inform future breakthroughs.

Each of the above exists in the context of a wider concern:

- **As a** person living in a world with ever more complex and numerous AI agents, **I want to** know whether their behaviour and learning are aligned with my best interests and those of my community, **so that** I maintain an ability to control them.

Motivated by these concrete use cases and overarching ethical imperative, there is growing interest in research into AI *interpretability*.¹ The ultimate goal of interpretability work is to equip human stakeholders with robust mental models of the learning and decision-making of AI agents. The particular way in which this is achieved must be guided by the prior knowledge of the target human, as well as the downstream application for which they intend to use the interpretation. As such, a vast array of techniques have been proposed, ranging from constraining agents to have simple rule-based architectures to post hoc analysis tools that estimate the causal effects of input conditions on an existing agent’s outputs. While this proliferation of perspectives may appear disjointed to those who are new to the field, it may also be necessary given the multifaceted and hard-to-formalise nature of the interpretability problem.

That said, the majority of existing interpretability methods are limited in scope, in that they focus on agents performing simple, one-step interactions with static datasets to solve prediction objectives (commonly known as *supervised* learning). Methods designed to interpret more general agents, whose interactions with their environments are inherently dynamic, are often transplanted from the supervised learning context with minimal changes, thereby failing to capture what makes these agents so challenging to understand in the first place. Relatively few approaches exist that face this challenge head-on.

This thesis contributes a suite of novel methods to the interpretability toolkit, which recognises the full complexity of the dynamic interaction between agents, environments and objectives. From one perspective, the methods are diverse, as each focuses on a different aspect

¹While this is our favoured term in this thesis, others have been coined with similar meanings, including “explicability”, “legibility”, “transparency”, and most commonly, “explainability” (with the associated “XAI” acronym denoting explainable AI). Some authors have tried to rigidly define these terms (e.g. [46, 81, 234]), but others continue to use them more fluidly. We do not wish to take a strong position in this terminological debate, since none of the terms perfectly capture the scope of the challenge at hand. However, the psychological literature contains thorough examinations of the nature of explanation, which frequently frame it as a fundamentally local activity of identifying aspects of a particular context (i.e. environment state) that cause or influence a particular observed phenomenon (i.e. agent action) [115]. Since human understanding of AI agents could be delivered by means other than this, such as simplified descriptions and visualisations of global behaviour [193], we find it beneficial to consider interpretability as the broadest umbrella concept, with explainability as a special case.

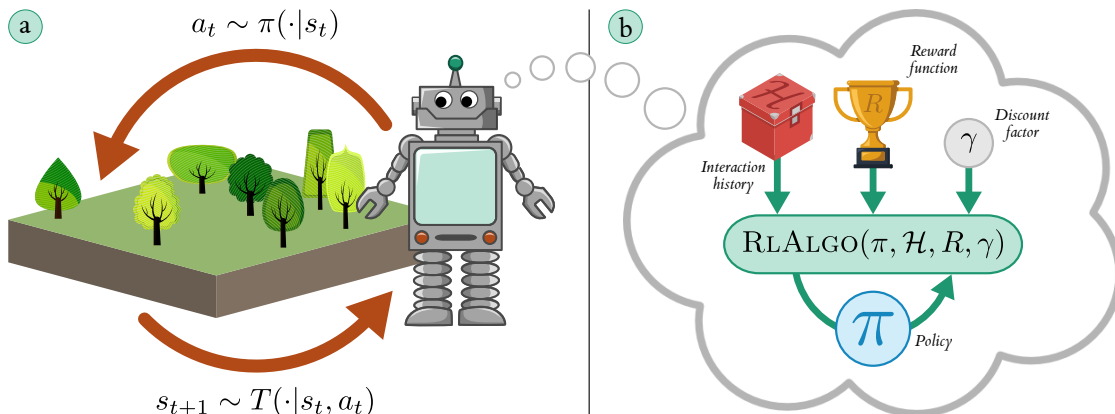


Figure 1.1: The agent-environment interaction cycle and generic reinforcement learning process.

of this interaction. From another perspective, however, the methods are all unified, because they build upon a common principle of abstraction with hierarchical rule-based models called *trees*. This decision to use tree models throughout provides a simple but expressive descriptive language that scales even to complex, high-dimensional environments, and enables the transfer of techniques for analysis and visualisation across chapters. Before further motivating this approach, this introductory chapter surveys prior approaches that have been taken to the agent interpretability problem. In order to ground this discussion in a standard set of terms, we first formalise our notion of an agent, which will be used throughout this thesis.

1.2 Agents and their Environments

The general agent paradigm is characterised by sequential interactions with an external environment over discrete-time intervals. As its input at time $t \in \mathbb{N}_0$, an agent receives information about the environment’s state $s_t \in \mathcal{S}$ and outputs an action $a_t \in \mathcal{A}$. The state then evolves according to a *dynamics* function $T : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$,² which is a conditional distribution over successor states $s_{t+1} \in \mathcal{S}$ given s_t and a_t . This cycle repeats as time advances. The environment may be *episodic*, meaning that it occasionally terminates and resets in a new state after entering a particular subset of terminal states and/or after a fixed time horizon $t = H$. Throughout this thesis, we assume that the agent acts according to a memoryless *policy* function $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, which specifies a distribution over actions to take in the current state, independently of all previous ones.³ Figure 1.1 (a) depicts the agent-environment interaction cycle.

We say that an agent is *learning* if its policy changes over time. In line with the generic machine learning paradigm outlined in the previous section, policy updates involve applying an optimisation algorithm to the parameters of the policy (possibly alongside those of other internal

²The notation $\Delta(\cdot)$ is used throughout this thesis to denote the set of all probability distributions over a set.

³This definition assumes that the agent’s observations uniquely identify the current state at each timestep. The more general class of *partially* observed environments is not considered in this work.

models) in order to increase the agent’s performance under some objective function. Although not all of the methods described in this thesis rely on this assumption, the canonical way of formulating an agent’s objective is via a *reward* function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, which outputs a scalar reward r_t conditional on the current state s_t and previous state-action pair (s_{t-1}, a_{t-1}) , alongside a discount factor $\gamma \in [0, 1]$, which specifies how the agent should exponentially discount rewards received further into the future. Given these additional concepts, we can define the *value* of each state $s \in \mathcal{S}$ under policy π as the expected discounted sum of future reward that an agent would obtain by following π starting in s . The value function has a recursive decomposition known as the Bellman expectation equation:

$$(1.1) \quad \begin{aligned} V_\pi(s) &= \mathbb{E}_{a \sim \pi(\cdot|s)} \mathbb{E}_{s' \sim T(\cdot|s,a)} \left[R(s, a, s') + \gamma \left[\mathbb{E}_{a' \sim \pi(\cdot|s')} \mathbb{E}_{s'' \sim T(\cdot|s',a')} [R(s', a', s'') + \gamma [\dots]] \right] \right] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s, a) [R(s, a, s') + \gamma V_\pi(s')], \end{aligned}$$

where “...” is a placeholder denoting recursion through all future actions and states (i.e. a'', s''', a''', s'''' , ...). Together, the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ defines a *Markov decision process* (MDP).

In an MDP, the agent’s objective is to find a policy π that maximises the value function V_π in every state. The agent is said to be performing *reinforcement learning* (RL) if it makes changes to its policy (1) with the aim of increasing V_π and (2) based on the history \mathcal{H} of states and actions generated during past agent-environment interactions, and their associated rewards. Figure 1.1 (b) gives a generic picture of the flow of information as an agent learns by RL. A diverse range of RL algorithms exists, and can be broadly taxonomised as follows:

- **Value-based:** use variants of the Bellman equation to learn an explicit estimate of the value function (or more commonly, the value for each state-action pair, known as the Q-function $Q_\pi(s, a)$) from interaction data. Construct a policy so as to incrementally improve estimated value, and iterate.
- **Policy-based:** bypass explicit value estimation, and instead directly optimise a parameterised policy function using historic rewards received from the environment.
- **Model-based:** perform supervised learning on (state, action, next state) transitions to learn an approximate dynamics model $T' \approx T$, and combine this with R and γ to forecast the effect of future actions. Construct an implicit policy by planning over some horizon.
- **Hybrid:** apply some synergistic combination of the preceding three methods, such as using value estimates to accelerate and stabilise parameterised policy learning (the *actor-critic* method), or using a learnt dynamics model to reduce the sample complexity of long-term value estimation. Most modern RL algorithms are hybrids to some extent.

Since this thesis does not concern the development of core RL algorithms, and since the majority of interpretability methods we present are purposefully agnostic to the underlying agent implementation, we will not go into any further detail on specific RL methods here. Relevant features of specific algorithms will be discussed as required in later chapters.

1.3 The Agent Interpretability Landscape

For simple MDP environments with small discrete state spaces, there exist guarantees that well-written RL algorithms will converge to optimal (i.e. value-maximising) policies [162], and it may be possible to describe these policies exhaustively in a form that humans can comprehend. In general, however, the flexibility of the agent paradigm can give rise to very complex learning problems, unstable optimisation processes, and nontrivial final policies that depend in subtle ways on the distribution of states encountered during learning. There is little hope of a human (especially a non-expert) understanding such a system unaided. In this section, we briefly survey prior efforts to render aspects of agents’ behaviour and learning more interpretable, in order to motivate and situate our own proposals. First, however, we review some basic concepts in interpretability and describe challenges presented by the dynamic agent context.

1.3.1 Defining Interpretability

While there is a broad consensus that interpretability is an important pillar of AI research, there is no widely-accepted definition of the term [161]. It may be best understood as a ‘squishy’ problem [237], of the kind more often encountered in the social sciences [177] than engineering and computing, and for which formal models can only provide approximate surrogates or partial perspectives. In part, this ‘squishiness’ derives from the contingency of interpretability on contextual factors, such as the algorithm and objective used by the target AI agent, the prior knowledge and skills of the human user [229], the specific queries they wish to have answered [233], and the downstream application to which they wish to put the interpretation [33].

This matrix of contingencies, alongside the lack of broadly-applicable quantitative metrics [39], means that evaluating proposed interpretability methods is notoriously hard [74]. Ultimately, the only complete end-to-end evaluation of an interpretability method may be to run a randomised controlled trial of users with and without access to the method, measuring success at the downstream application. This has been done in some narrow contexts [197, 198], but is often impractical. Instead, it is typical to resort to heuristics about what an interpretable system looks like, and assess proposed methods based on their adherence to these heuristics.

In that vein, it seems likely that interpretability is related to the notion of *simplicity*. Humans find systems hard to understand if they deem them to be complicated. Given a complicated system, such as the dynamic interaction between AI agent, environment and objective, following this heuristic leads us to consider either modifying the system to enforce simplicity (the so-called *intrinsic* perspective on interpretability) or building simpler models that approximate it from the outside (the *post hoc* perspective). Of course, this defers the conceptual difficulty onto defining simplicity, which is no easy feat [18, 205], but further heuristics present themselves. Notwithstanding the dependency on human users and their expertise, a technical system may be made (or modelled as) simpler by rendering nonlinear things linear, continuous things discrete,

many things few, and interacting things independent. Furthermore, the constituent parts of the system must be grounded in yet simpler base concepts, which in part means that they can be expressed as brief statements of natural language. Hence, an ideally interpretable system may consist of a few discrete entities, interacting in a small number of discrete ways, where both the ‘entities’ and the ‘ways’ correspond to short natural language expressions. To reiterate, in the context of AI, such an outcome can be achieved either through intrinsic modifications to the agent, environment or objective themselves, or by post hoc approximate modelling.

These heuristics capture a significant proportion of existing interpretability methods, as well as our own approach. As outlined in the next chapter, our general strategy is to build simple, discretised representations of AI agents, grounded in rules expressed in natural language, through a generic process called tree abstraction. Although we mostly take a post hoc perspective, later chapters move in an intrinsic direction by using trees as interpretable reward functions for RL.

1.3.2 Challenges of the Agent Context

As discussed above, the bulk of work on AI interpretability focuses on supervised learning systems. These are characterised by a single function $f : \mathcal{I} \rightarrow \Delta(\mathcal{O})$ that maps points in an input space \mathcal{I} to (a distribution of) points in an output space \mathcal{O} . Most interpretability methods thus amount to enforcing (intrinsic) or approximating (post hoc) simplicity in that function f . On the intrinsic side, interventions are made to increase the degree of linearity, discretisation or decomposability in the internal structure of f [180], or enforce grounding to simple natural language concepts [147]. Post hoc methods divide into *local* approximations, which often work by identifying features of the input space that heavily influence [166, 208] or change [261] a particular output, and *global* ones, which construct a simplified, often rule-based, surrogate model to approximate the original function across many or all inputs [180].

In the dynamic agent context formalised in Section 1.2, far more is at play than a single function. The closest analogue to f is the agent’s policy π , which maps states to actions, and it is common to see methods for interpreting f carried over directly. While valuable insight can be gained from such an approach, that insight is inherently partial. This is because agents do not act on independent and identically distributed (i.i.d.) inputs, but on environment states that depend inescapably on their previous actions. This has important consequences for the types of qualitative phenomena that agent-environment interaction can produce. For instance:

- The **behaviour** of an agent can only be fully understood at the level of many consecutive timesteps of states and actions (called *trajectories*). Emergent features at the trajectory level cannot readily be predicted from an input-output analysis of single timesteps.
- The **performance** of an agent is also aggregated over extended trajectories rather than single actions. In the case of an MDP, performance is measured by the value function, which is the expected discounted sum of per-timestep rewards.

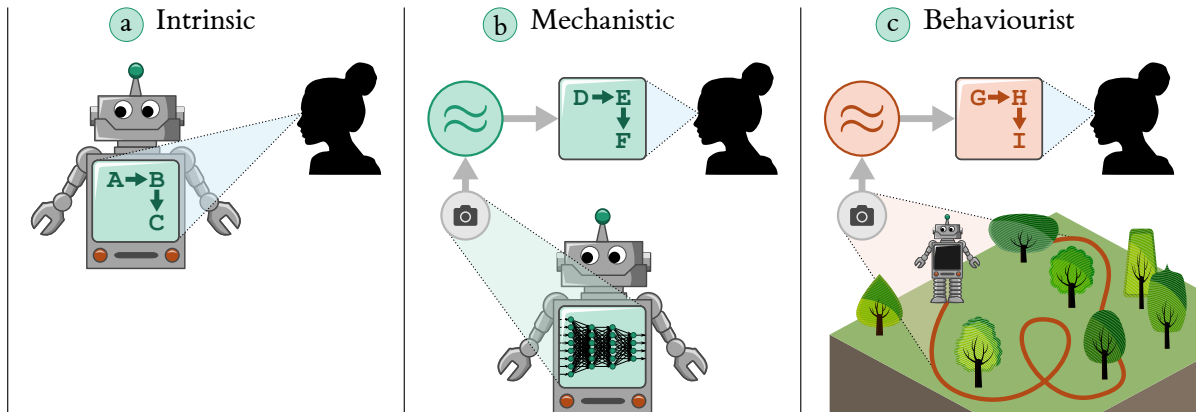


Figure 1.2: The three perspectives on agent interpretability.

- The **learning** of an agent, especially by RL, depends on state-action data generated during its own past interactions with the environment. This results in a cyclic dependency of future performance on past policies, which makes understanding the dynamics of agent learning just as important as understanding the final policy.

Each of these phenomena arises from interactions between the agent policy π , environment dynamics T and objective (e.g. reward function R). Accordingly, an interpretability researcher is presented with many possible intervention points at which they could enforce or approximate simplicity. When surveying existing methods, prior review papers have applied a variety of taxonomies to understand this diversity of possible approaches [98, 175, 199]. In the following subsections, we continue to emphasise the distinction between intrinsic and post hoc perspectives. However, we propose to further subdivide post hoc methods into those that focus on the internal structure of an agent (e.g. its policy model and constituent parameters) and those that focus on its external behaviour and performance in the environment, and how they play out over time. We find this distinction to be conceptually valuable, and note a parallel with the division between the cognitivist and behaviourist paradigms in psychology [43]. This leads to the following taxonomy of agent interpretability methods:

- **Intrinsic:** Methods that enforce interpretable structure within the agent itself, so that a human can inspect and comprehend its functionality directly (see Figure 1.2 (a)).
- **Mechanistic:** Methods that examine an existing agent’s internal mechanisms to derive human-interpretable insights. As indicated by the “ \approx ” in Figure 1.2 (b), this almost always requires some simplifying approximations to be made.
- **Behaviourist:** Methods that examine an existing agent’s external behaviour to derive human-interpretable insights (also via approximation; see Figure 1.2 (c)).

We now follow this taxonomy to briefly survey prior work on agent interpretability, commenting upon points of connection to our own proposals.

1.3.3 Intrinsic Methods

Intrinsic methods embody a ‘designer’s approach’ to the interpretability problem: constrain the structure of the agent itself to make it easier for a human to study. The central structure of concern is the agent’s policy function $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, which in modern work is often implemented as a densely connected deep neural network [240]. A natural direction is thus to replace that network with something simpler and more modular, such as a collection of human-readable symbolic rules [65], a selection over a finite set of domain-specific behaviour programs [260], or a closed-form algebraic expression assembled by genetic programming [78, 113].

Of particular relevance to this thesis are the many prior attempts to use tree-structured policy functions [201, 213, 252]. While we formalise the notion of a tree in the next chapter, it currently suffices to say that it is a hierarchical structure of rules expressed over features of the environment state, which determine which of several predictions (e.g. actions) is made. As long as the number of rules is not too large, tree-structured policies are generally seen as interpretable because their local and global functionality can be represented by textual statements or diagrams. Extensions of the basic model, which allow more complex multi-feature rules [222], rules with probabilistic outcomes [230], or predictions that have the form of linear models [163], enable more expressive policies, but arguably hamper interpretability.

Structured policy models usually require a predefined set of interpretable state features to use in rules or algebraic expressions. [93] develop a pipeline for deriving such features from data, and [73] use a second tree model to learn a linear transformation of the state space for inputting to the main policy tree. Other work on interpretable state representations includes relational RL [79], which describes environments in terms of discrete objects and their interactions, and various agent models that base their policies on a small number of prototype states [121] which may be specified manually [140]. For environments whose states are represented as images (e.g. video games), attention mechanisms have been used to equip agents with a ‘gaze’ that reveals which visual features are being attended to [267], although there is evidence that this mechanism picks up on surface-level patterns rather than meaningful game structures [10]. Pretrained language models have also been used to assign textual labels to image observations, equipping an agent with a compressed and interpretable memory of its past inputs [194].

On the opposite end of the input-output pipeline are methods for agents to produce richer and more revealing outputs instead of unjustified actions. For example, [134] modify a popular value-based RL algorithm to predict not a single scalar value for each state-action pair, but a *decomposed* value vector, indicating the influence of various reasons for selecting one action over another. In a similar spirit of decomposition, [269] learn a model that predicts future visitation to every other state-action pair in the environment, which (via a dot product with a vector of rewards) recovers the value function. This enables the most influential future events to be identified as reasons for action. For environments with pass/fail objectives, [54] transform an agent’s value predictions to correspond to the predicted probability of success, which is deemed

to be more interpretable by non-expert humans. Hand-crafted natural language templates have been used to allow an agent to describe the reasoning behind a single action or trends in its global policy [110, 253, 263]. Finally, models have been proposed for how an agent should modify its behaviour to make it more transparent about what objective is being pursued [23, 123].

The intrinsic perspective on AI interpretability has vocal supporters [77, 214] and is certainly a valuable research direction. However, it may be the hardest to advocate for, given the prevailing trend towards increasingly complex and unspecialised learning agents based on neural networks. Although some arguments have been made to the contrary [225], it is often observed that enforcing intrinsic interpretability sacrifices quantitative performance [19]. Since many of the above methods rely on domain-specific assumptions or knowledge, such as plausible behaviour programs [260], object-oriented representations [79], representative prototype states [140], or semantically decomposable reward functions [134], their generality is quite limited. For better or worse, this limits their appeal to a mainstream AI community with an increasing aversion to hand-specified priors and constraints [239]. The majority of work in this thesis does not adopt the intrinsic perspective, and instead constructs post hoc models of existing agents and their environments based on limited (or no) assumptions about their internal structure.

1.3.4 Mechanistic Methods

In contrast to the intrinsic perspective, work on mechanistic interpretability embodies a ‘natural scientist’s approach’: taking the internal structure of an agent as given, develop techniques for probing that structure to gain partial insights into its operation. An example is the analysis of *feature importance* (also influence/attribution), which measures the sensitivity of an agent’s response to each feature of a given environment state, often by applying small perturbations. The ‘response’ can be defined in many ways, including the magnitude of a continuous-valued action output by the policy [111], the value estimates of different actions in value-based RL [200], or the output of a learnt reward function [169, 216]. Some of the most principled feature importance methods are based on the game-theoretic Shapley value framework [166, 227]. Various ways in which Shapley values can be used to interpret RL agents are surveyed in [22].

For environments with image observations, the term *saliency* is used to denote the feature importance of pixels or image regions, which are often overlaid as heatmaps [125]. The intuitiveness of this visualisation format allows it to be animated or extended into 3D to understand how a policy’s saliency map evolves over the course of a behavioural episode [75] or from one point in learning to another [100]. However, the interpretation and evaluation of saliency maps are often highly subjective, so it is advisable to use them with caution [14].

Somewhat similar, and theoretically connected [3, 148], to feature importance analysis is the literature on *counterfactual explanation*, which uses an (often learnt) generative model to perturb state features until an agent’s action changes in a prescribed direction. Most existing methods work with visual state representations, in which case the counterfactual can be shown as an image [124, 190]. Counterfactuals are widely favoured on the premise that grounding

reasons for an output in an explicit contrast case is natural and human-like [177]. It also tends to be preferred by recipients in practice [54]. Slightly adjacent to this work are methods that search for ‘interesting’ or ‘critical’ states that maximise a particular agent output, such as the estimated value of a target action [215], or the entropy (i.e. uncertainty) of the policy [122].

Both feature importance and counterfactual explanation focus on the input-output behaviour of an agent or one of its submodules. Other methods go deeper to directly inspect the agent’s internal representations. For example, work has been done to visualise the features detected by particular neurons in a value function network [116], or find a low-dimensional embedding of network activations across many states, enabling the discovery of clusters and prototype states [272]. For agents with recurrent policy networks for partially observed environments, several works have visualised the dynamic memory vector and how it correlates with environment state features over time [112, 129]. The dynamics of this vector are complex and high-dimensional, leading others to try approximating it as a finite state machine [149].

Each of the above methods provides useful but narrow insight into agent mechanisms. For this reason, it is common to take a ‘toolbox’ approach to mechanistic interpretability, in which several methods are combined into a single analysis and visualisation pipeline [116, 129, 168]. A similar combination has also been used to understand reward functions [174]. Perhaps the most complete example of the toolbox approach is an extended analysis of the AlphaZero agent as it learns to play chess [172]. This analysis makes heavy use of domain-specific concepts (i.e. common chess piece layouts), which are tested for presence in the agent’s internal networks.

The mechanistic perspective on AI interpretability captures a diverse body of work [40]. It provides insight into an agent’s low-level operation, without the drawback of performance-limiting constraints on the agent itself. However, for complex agent architectures, it is hard to obtain global understanding without many simplifying assumptions. Mechanistic methods are also often tailored to specific agent or environment types, which limits their breadth. Above all, and with few exceptions (e.g. [149]), most methods draw directly from the supervised learning literature, so operate on isolated agent responses to single environment states. Issues of dynamics are either addressed by repeating a method many times [100] or ignored altogether [190]. Although we revisit the concept of counterfactual explanation, most work in this thesis does not adopt the mechanistic perspective. Instead, we primarily develop behaviourist models, which are natively dynamic and agnostic to agent and environment details.

1.3.5 Behaviourist Methods

Behaviourist methods retain the narrative of interpretability as a natural science, but focus on an agent’s behaviour as realised in the environment, rather than its mechanism or function as an isolated input-output system. In this sense, the approach is less like cognitive neuroscience, and more akin to animal *ethology* [196, 248]. Behaviourist models are based on external observations of the agent’s interaction with its environment over time, with few assumptions about how its actions are produced. Some go as far as to treat it as a fully inscrutable *black box*.

A popular approach is *policy modelling*, which uses observations of an original agent to learn a simpler, more interpretable policy model that replicates its behaviour or performance closely as possible. Through scrutinising, or even formally verifying [20], the interpretable model, the aim is to establish appropriate trust and understanding of the original policy. Although policy modelling has been done using a variety of rule-based architectures [52, 186], trees are by far the most popular choice. As in the literature on intrinsically interpretable policies, basic tree models consist of single-feature rules leading to single-action predictions [20], but this can be extended to probabilistic [51, 251] or nonlinear [71] rules, linear model predictions [97], or a mixture of multiple ‘expert’ trees [256]. Post hoc tree models have also been proposed for various other tasks, including inferring the goal being pursued by a black box agent [34] and classifying safe and unsafe regions of the state space for a given policy [221].

In high-dimensional environments, trajectories of agent behaviour are hard to visualise directly. For this reason, several works have explored discretising the state space into regions called *abstract states*, and summarising trajectory data as a graph of transition probabilities between them [60, 171, 209, 249]. Abstract states can be defined as areas where the agent’s action or value function are similar [171, 249], or simply by spatial proximity [60]. In image-based environments, an abstraction can be found by clustering in a low-dimensional embedding space, and representing each abstract state by its mean image [209]. [171] provide semantic grounding for abstract states, thereby easing the interpretation of transition graphs, by constructing them using user-specified predicates. Although described differently, the *aggregated trajectories* approach used in [1] has a similar effect of reducing a large amount of continuous trajectory data to a graph over discrete representative points.

Instead of building a statistical model of agent behaviour, an alternative way to summarise it is through a small set of exemplary state-action pairs or trajectories. [7] surveys aspects of this approach, which include the intelligent selection of examples and the development of intuitive interfaces for presenting them to a human (short videos are common). Various methods base their selection criteria on (in)frequency of occurrence [226], diversity relative to other selected samples [6, 59], or a notion of *criticality* with respect to future reward [6, 105]. Other works have used trajectory examples to illustrate the best, worst and most likely outcomes after a particular state [224] or highlight points of disagreement between two agents’ policies [8]. [66] seek to identify which of many trajectories encountered during an agent’s learning contributes most to its current policy. [9] use user-specified logical queries to identify trajectories of interest.

The counterfactual explanation methods in the preceding subsection focus on an agent’s immediate response to the current state, but other counterfactual problems can be posed that account for the stochastic and sequential nature of agent-environment interaction [90]. For example, given an agent trajectory, [250] search for small perturbations to the sequence of actions taken that would have led to higher reward. At a more global level, both [67] and [184] learn a minimally-perturbed policy that alters an agent’s total expected reward (or value) by a

prescribed amount, and visualise the results with paired trajectory examples. Alternatively, [91] revisit the standard framing of state perturbation to realise a counterfactual action, but add a constraint that the new state be reachable from the original one given the environment dynamics. Finally, [87] consider making targeted modifications to the environment itself until an agent learns a policy that conforms to a user’s expectations.

Some behaviourist interpretability work fits the label of *visual analytics*, in that it involves gathering large amounts of agent-environment interaction data and presenting it in a human-digestible form via interactive images and plots [2, 68, 112, 173, 179, 223, 262]. The plots tend to span several spatiotemporal scales, from individual states, through extended trajectories and clusters of related trajectories, to the entire timeline of agent learning. Given the large amount of information carried by such visual analytics dashboards, their stated audience tends to be expert AI practitioners, interested in debugging agents [68] or environment implementations [173].

The behaviourist perspective on AI interpretability is perhaps the most dissimilar to work in the supervised learning domain. Because it focuses on external behaviour, it is not a route to fine-grained insight into agents’ internal mechanisms. However, for the same reason, it places its emphasis at the level in which many human stakeholders are interested, namely what an agent *actually does* when deployed in its environment, and the effect that has on the objective and environment itself. Most behaviourist methods have the advantage of being model-agnostic, so can be used on existing agents today while being robust to future architectural changes that may render mechanistic methods impractical or ineffective. Since the behaviourist perspective recognises the complexity of dynamic agent behaviour, the potential diversity of methods is huge and far from fully explored. This thesis exploits this opportunity. While the first half of Chapter 3 presents and analyses a model that resembles prior policy modelling work, we soon diverge from this precedent to capture other dynamic aspects of agent behaviour and learning.

1.4 Themes, Gaps and Opportunities

Based on the above, we identify seven cross-cutting themes in agent interpretability work:

- **T1: Simplicity.** Some of the simplest, most generic methods are also the most cited (e.g. value decomposition [134], critical trajectories [6]). This may be because their purpose and assumptions are clear, and they are easy for others to build upon.
- **T2: Examples.** Although much interpretability work is concerned with building abstract statistical models, specific examples (e.g. prototype states [272], influential trajectories [66]) provide useful grounding, especially for non-experts.
- **T3: Contrasts.** As has been noted before [177], many interpretability methods involve summarising points of difference between alternative cases. This not only includes counterfactual explanation [90], but other methods too (e.g. policy disagreements [8]).

- **T4: Interactivity.** There are calls for AI interpretation to be responsive to specific user needs and queries [233], and several methods deliver on this (e.g. logic-based trajectory queries [9], user-expected policies [87], interactive plots [68]).
- **T5: Presentation.** Many modes of visual or textual presentation have been used to convey information extracted by interpretability methods. Presentation tangibly affects how humans receive interpretations, and the wrong choice can cause information overload [197].
- **T6: Combination.** Because post hoc interpretation is inherently partial and approximate, it is common to combine complementary methods to gain more holistic insight (e.g. mechanistic interpretability toolboxes [172], visual analytics dashboards [262]).
- **T7: Trees.** Most concretely, there is a clear precedent of tree models being used from both intrinsic and post hoc perspectives. Although this is mostly as policies [252] or models thereof [20], other uses have been explored (e.g. goal recognition [34]).

We also identify six common gaps and shortcomings of existing approaches, which create opportunities for novel contributions in our own work:

- **G1: Dynamics.** Due to their origins in the supervised learning space, many methods treat agents as isolated input-output systems, ignoring temporal information. Others have begun to acknowledge and address this issue, but it remains underrepresented.
- **G2: Learning.** Most methods analyse a single, fixed agent policy (e.g. the final one produced by RL). Few can represent the changes that occur as an agent learns over time, which are crucial for understanding the provenance and capabilities of the final policy.
- **G3: Generality.** It is common for methods to be specialised to certain agent algorithms (e.g. value-based RL), or certain classes of environment or task. More application-agnostic methods are more likely to see wide adoption.
- **G4: Unification.** Existing methods are diverse and disunified. Although this is a product of healthy exploration, there would be value in unified frameworks for representing multiple aspects of agents and environments in common language and visualisations.
- **G5: Contextualisation.** The issue of interpretability is often tackled in isolation, without considering the context in which it is needed, or how it integrates with other open problems in AI research, such as teaching agents to reliably satisfy human preferences.
- **G6: Application.** Methods tend to be deployed on well-behaved benchmarks (e.g. Atari games, classic control problems). Although this aids understanding and comparison, it is also beneficial to motivate and evaluate them in more realistic industrial use cases.

1.5 Thesis Contributions

This thesis contributes a selection of novel perspectives and models for agent interpretability, which align with the themes identified in the previous section, while addressing some of the gaps. Together, they go some way to providing a holistic understanding of an agent’s behaviour, by focusing in turn on its policy, value function, state dynamics, learning progression and reward function. At the same time, they maintain methodological coherence, because they are all based on tree-structured models. This work is organised into the following chapters. Connections to the existing research themes (**T**) and gaps (**G**) are highlighted where relevant:

- Chapter 2 introduces tree models (**T7**) as a common framework to be used throughout this thesis (**G4**), and discusses how they embody desiderata for interpretable representations. We present trees as a general tool for abstracting any complex system (**G3**), that can be constructed in various ways to answer different queries that a human may have (**T4**), and can be visualised either as diagrams or geometric structures (**T5**).
- Chapter 3 presents two behaviourist tree models of agents with fixed policies. In introducing the first, which is a basic policy model, we outline the simple and standardised strategy by which trees are learnt throughout this thesis (**T1**). After a quantitative evaluation, we show how the model enables the counterfactual explanation of agent actions (**T3**), and the summary of decisions made during an extended behavioural episode (**G1**). The second method extends the first by combining additional information (**T6**) about an agent’s value function and state dynamics (**G1**). We propose novel visualisations of the information contained in this multiattribute tree (**T5**), and use the model to generate hypothetical agent trajectories (**T2**) based on user-specified queries (**T4**).
- Chapter 4 uses a tree to model how an environment’s state evolves under a given agent policy (**G1**), and how this changes as the agent learns over time (**G2**). It is specifically optimised to highlight contrasts between different time points (**T3**). We visualise this state transition model in a variety of ways, including with graphs and heatmaps (**T5**), and show how prototype trajectories can be identified to summarise an agent’s behaviour at each stage of learning (**T2**). This model’s extreme generality means it can be applied to any learning agent, or even other nonstationary dynamical systems (**G3**).
- Chapter 5 shifts perspective to explore how trees can provide intrinsic interpretability by serving as an RL agent’s reward function. Furthermore, we show how reward trees can be learnt from human feedback (**T4**), thereby drawing a connection between interpretability and the literature on human-agent alignment (**G5**). The efficacy of reward tree learning is established via experiments with synthetic and real human feedback on four benchmark tasks. We then demonstrate interpretability, showing how the use of a reward tree enables detailed monitoring of an agent’s learning progress (**G2**), and how visualisations developed earlier in the thesis can be repurposed for this new tree model (**G4**).

- Chapter 6 applies reward tree learning to a realistic aviation use case (**G6**), finding it to be competitive with neural networks. The method is refined to simplify preference collection (**T1**), unify with existing literature (**G4**), and grow the tree differently to improve performance. Because the method is agnostic to the agent algorithm (**G3**), we are able to switch to using model-based RL, which accelerates learning. New interpretability techniques are developed to attribute reward predictions back to individual preferences (**T2**) and explain the changes that occur as a model-based agent plans its action (**T3**).
- Chapter 7 concludes by discussing the strengths, limitations, and potential for combining (**T6**), the proposed methods. It also considers some directions for further work.

Taken as a whole, the positioning of this thesis may be best understood through the parable of *The Blind Men and the Elephant*, depicted in Figure 1.3.⁴ In the parable, a group of blind people attempt to apprehend the form of an elephant by touching its various body parts. Each gains only partial insight in isolation, but a more complete picture emerges if the individuals can communicate their respective findings. Given that any method for interpreting AI agents has inherent limitations and assumptions, a similar integration of approaches is necessary to provide a holistic understanding. This integration is made easier by expressing all models in a common language, and this is what we seek to do through our consistent use of trees.



Figure 1.3: The parable of *The Blind Men and the Elephant*.

⁴Image from https://commons.wikimedia.org/wiki/File:Blind_monks_examining_an_elephant.jpg. This parable has been invoked in a prior review of interpretability for supervised learning [234], but we see it as even more relevant to agents in dynamic environments.

1.6 Research Output

The following publications were produced during the course of this PhD:

- **Tom Bewley**, Jonathan Lawry, and Arthur Richards. “Modelling Agent Policies with Interpretable Imitation Learning.” Published in Trustworthy AI - Integrating Learning, Optimization and Reasoning, Springer LNCS, 2021.
 - This paper is the basis of the first half of Chapter 3.
- **Tom Bewley** and Jonathan Lawry. “TripleTree: A Versatile Interpretable Representation of Black Box Agents and their Environments.” Published at AAAI Conference on Artificial Intelligence, 2021.
 - This paper is the basis of the second half of Chapter 3.
- **Tom Bewley**, Jonathan Lawry, and Arthur Richards. “Summarising and Comparing Agent Dynamics with Contrastive Spatiotemporal Abstraction.” Presented at IJCAI Workshop on Explainable Artificial Intelligence, 2022.
 - This paper is the basis of Chapter 4.
- **Tom Bewley** and Freddy Lecue. “Interpretable Preference-based Reinforcement Learning with Tree-Structured Reward Functions.” Published at International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2022.
 - This paper is the basis of Chapter 5.
- Joseph Early*, **Tom Bewley***, Christine Evers, and Sarvapali Ramchurn. “Non-Markovian Reward Modelling from Trajectory Labels via Interpretable Multiple Instance Learning.” Published at Conference on Neural Information Processing Systems (NeurIPS), 2022.
 - This *joint first-author paper is not discussed in this thesis.
- **Tom Bewley**, Jonathan Lawry, and Arthur Richards. “Learning Interpretable Models of Aircraft Handling Behaviour by Reinforcement Learning from Human Feedback.” Accepted for presentation at AIAA SciTech Conference, 2024.
 - This paper is the basis of Chapter 6.
- Scott Jeen and **Tom Bewley**. “Conservative World Models.” Under review.
 - This paper is not discussed in this thesis.

Chapter 2

Abstraction with Trees

2.1 Introduction

The aim of AI interpretability work is to communicate information about agent decision-making, behaviour and learning to humans. In doing so, we must decide upon a representational framework, or *lingua franca*, in which to express that communication. In this thesis, we use tree models as our representational framework. This chapter provides a general discussion of trees, and introduces mathematical notation which unifies the tree-based methods presented hereafter. We also seek to justify our a priori judgement that using trees is a promising approach by showing how they satisfy several desiderata for human-interpretable representations.

For the sake of maximum generality, most of this chapter is presented as a context-agnostic discussion of tree abstraction, and our motivating application of agents is only revisited at the end. From the next chapter onwards, all work is firmly grounded in the agent context.

2.2 Symbols and Abstractions


In a recent paper, Kambhampati et al. argue that communicating information between a human and an artificial agent in terms of the raw data processed by modern machine learning models (typically high-dimensional numerical tensors representing temporal sequences of states, actions and rewards) is liable to place an “*intolerably high cognitive load*” on the human [137]. They advocate for the development of *symbolic interfaces* through which both agent-to-human explanations and human-to-agent advice can be exchanged. This proposal to use discrete symbols as the *lingua franca* for human-agent interaction is well supported by theories and empirical findings in cognitive science. Much of human reasoning is conducted in natural language, which is universally discrete [117], so constraining agent communication to have a symbolic form may make it easier for the human to find a mapping to and from their native representations. However, this requires us to address a challenge at the other end of the communication pipeline, regarding how symbols can be derived from the raw data representations used natively by

agents. This ubiquitous data-to-symbol gap-bridging dilemma has a long history in AI research, and is known as the *symbol grounding problem* [109].

Our approach to the symbol grounding problem takes inspiration from the work of three influential figures from AI and cognitive science: Lotfi Zadeh, Douglas Hofstadter and Peter Gärdenfors. In Zadeh’s fuzzy set theory [271], Hofstadter’s analogy-making framework [118] and Gärdenfors’ conceptual spaces model [94] alike, symbols are grounded by grouping raw data instances into larger units based on a measure of similarity, proximity or functionality. This grouping operation is variously known as *coarse-graining*, *generalisation*, *categorisation*, *granulation*, *aggregation* and *abstraction*. In this thesis, we adopt the latter term. Abstraction is a fundamentally lossy process since detailed information about each raw data instance underlying an abstract symbol is overlooked when statements are made at the symbolic level. However, for a system that seeks to communicate with computationally-bounded humans, such losses may be worth the gain in comprehensibility through simplicity. The art and science of abstraction lie in preserving only the information deemed most important for a given downstream application. Zadeh, Hofstadter and Gärdenfors all contend that defining informative grounded symbols through judicious abstraction is central to human intelligence.

In the following sections, we build on these ideas to develop a general theory of abstractions satisfying various structural and semantic desiderata, and discuss how it can be applied to data generated by agents interacting with dynamic environments. We adopt the language of Gärdenfors’ conceptual spaces as our point of departure because Hofstadter’s framework is less mathematical in its presentation, and Zadeh’s departs in a nontrivial way from our own methods by focusing on *fuzzy* abstractions.¹

2.3 From Conceptual Spaces to Tree Abstractions

Gärdenfors defines a *conceptual space* as a collection of *quality dimensions* which characterise observations of a system. These dimensions can be grouped into *domains*, which are subsets of dimensions that have natural semantics that distinguish them from the others. Canonical examples of domains include the colour dimensions of hue, saturation and brightness, and the sound dimensions of pitch and loudness. A conceptual space is thus a collection of one or more domains. In this thesis, we consider conceptual spaces that have a domain of D real-valued dimensions, thus forming a real vector space \mathbb{R}^D . We refer to this domain as the *feature space*. Any other dimensions in the conceptual space are known as *attributes* \mathbb{Y} ; their structure will be revisited later. An example of a conceptual space in $(\mathbb{R}^D, \mathbb{Y})$ form is depicted in Figure 2.1 .

¹Zadeh’s symbols are fuzzy objects, which means that their membership functions in the raw data space are continuous-valued. For reasons of computational and syntactic simplicity, this thesis focuses on *crisp* abstraction techniques, in which each raw data instance is either definitely a member of a given abstract symbol, or definitely not a member. The potential for fuzzy variants of the methods developed is ripe for investigation in future work.

2.3.1 Convexity

Gärdenfors hypothesises that humans ground symbols in *convex subsets* of domains in conceptual spaces. Here, convexity refers to the property that for any two points in a subset x , all other points between them also lie in x , where the notion of *betweenness* is formalised differently depending on the topological structure of the domain. In the preceding examples, Gärdenfors’ hypothesis implies that the symbol “red” is grounded in a convex subset of the colour domain and the symbol “screech” is grounded in a convex subset of the sound domain. Let a *convex abstraction* be a set of m convex subsets of a domain, $\mathcal{X} = \{x_1, x_2, \dots, x_m\}$, each of which grounds a symbolic label. To complete our first example, the set of all colour labels “red”, “green”, “blue”, ... is grounded in a convex abstraction of the colour domain.

In a vector feature space with the Euclidean distance metric, the notion of betweenness, and thus of a convex subset, is well-defined. For two feature vectors $\mathbf{f}, \mathbf{f}' \in \mathbb{R}^D$, the set of points between them are those that lie on the intervening straight line segment, i.e. $\text{between}(\mathbf{f}, \mathbf{f}') = \{c\mathbf{f} + (1 - c)\mathbf{f}' : 0 \leq c \leq 1\}$. A subset $x \subseteq \mathbb{R}^D$ is thus convex if (and only if) for all pairs $\mathbf{f} \in x$, $\mathbf{f}' \in x$, $\text{between}(\mathbf{f}, \mathbf{f}') \subseteq x$. Figure 2.1 (b) shows three subsets of \mathbb{R}^D , where x_1 and x_2 are convex, but x_3 is not, and hence $\{x_1, x_2, x_3\}$ is *not* a convex abstraction.

2.3.2 Partitioning

Alongside convexity, Gärdenfors considers a second geometric desideratum for human-like abstractions: that they *partition* their domain. Using our feature space notation, an abstraction \mathcal{X} is a partition if (and only if) it covers the entire domain without gaps ($\bigcup_{x \in \mathcal{X}} x = \mathbb{R}^D$) or overlaps ($\bigcap_{x \in \mathcal{X}} x = \emptyset$). A partition has the simplifying property that every $\mathbf{f} \in \mathbb{R}^D$ belongs to exactly one $x \in \mathcal{X}$, and thus there is a unique symbolic label for every possible feature vector. While it may be questioned whether or not this property holds for natural language (there may well be gaps and overlaps in the grounding of English words in the visible colour domain), it is mathematically convenient. A partition also provides a complete model for domain-wide generalisation since it allows us to define a total function that maps from the continuous domain of features to the discrete domain of abstract subsets. The methods presented in this thesis rely on the existence of such a function.

Figure 2.1 (c) shows a convex partition of \mathbb{R}^D . A notable result is that to satisfy both the convexity and partitioning desiderata, the abstraction must consist entirely of polytopes, which are bounded by hyperplanes [156]. This is because, in a partition, every boundary of a subset meets the boundary of another subset. If that boundary were outwardly curved for one subset (e.g. the red dotted line for x_3), it would be inwardly curved for the other (in this case, x_4), violating the convexity requirement. Hence, the only possibility is for the boundary to have no curvature, which is the definition of a hyperplane.² A hyperplane in \mathbb{R}^D is described by a linear equation of the form $b_1\mathbf{f}_1 + b_2\mathbf{f}_2 + \dots + b_D\mathbf{f}_D = c$.

²A more formal proof of the polytope property is given in Section 3.1 of [156].

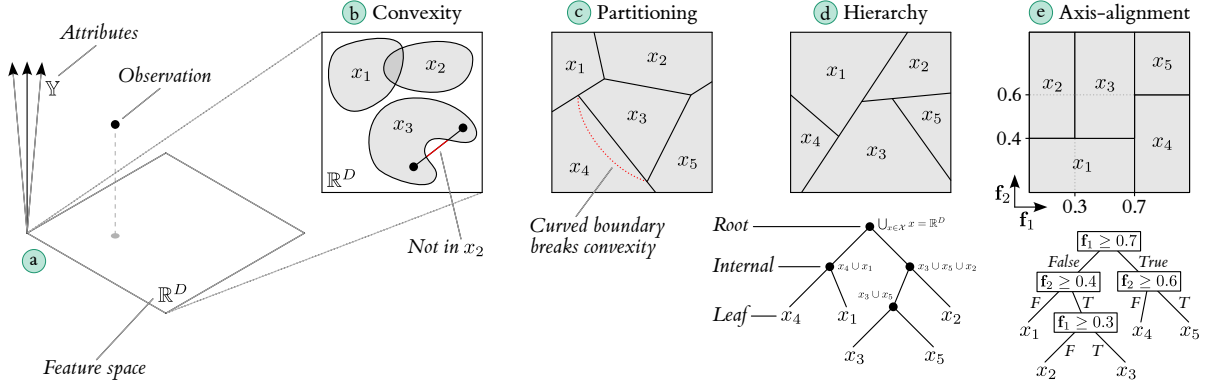


Figure 2.1: A conceptual space in $(\mathbb{R}^D, \mathbb{Y})$ form, and four geometric desiderata for abstractions.

Gärdenfors suggests *Voronoi tessellation* as an algorithm for generating convex partitions of real vector domains. Given a *prototype* point for each of the symbols, $\mathbf{f}^1, \dots, \mathbf{f}^m \in \mathbb{R}^D$ (e.g. in the colour domain, one each for “red”, “green”, “blue”, ...), Voronoi tessellation divides the space into polytopic regions that are closest to each prototype according to Euclidean distance. In addition to the biological plausibility of prototype theory as a mechanism for concept acquisition [210], one reason for favouring Voronoi tessellations over general convex partitions is their memory efficiency: they are fully defined by ‘only’ mD parameters specifying the D -dimensional position of each prototype. However, this still scales linearly with D , and the geometry and arrangement of the resultant high-dimensional polytopes do not follow straightforwardly from knowledge of the prototypes themselves (each polytope is not a local function of its own prototype, but of an entire neighbourhood of surroundings ones).

In light of these limitations, we now propose two additional desiderata for human-interpretable abstractions that go beyond those suggested by Gärdenfors, and lead to the tree abstraction model adopted in this thesis.

2.3.3 Hierarchy

It is widely accepted that human language and concept understanding is *hierarchical*, in that symbolic labels are often associated with special cases or generalisations of other symbolic labels [24]. Organising data hierarchically allows phenomena to be described and analysed at varying levels of granularity, from the generalised and simplistic to the specific and detailed. It thereby enables the management of a tradeoff between information preservation and representational compactness. This tradeoff is fundamental for any system that seeks to be intelligible to humans, and re-emerges several times throughout this thesis. The intuitive notion of hierarchy can be formalised in many ways. In the following, we present a definition that applies to convex partitions \mathcal{X} of a feature space \mathbb{R}^D , which allows us to begin talking about abstractions as being tree-structured.

Let us say that a convex partition \mathcal{X} can be *pruned* if there exists a pair $x, x' \in \mathcal{X} : x \neq x'$

such that the union $x \cup x'$ is itself a convex subset. In this case, there exists a more granular convex partition $\mathcal{X}' = (\mathcal{X} \setminus \{x, x'\}) \cup \{x \cup x'\}$, which has one fewer subset. Now suppose that \mathcal{X}' can also be pruned to create a yet more granular convex partition \mathcal{X}'' , and that the ability to prune persists recursively, such that it is possible to take the union of pairs of convex subsets until a single set covering the entire space is recovered. As shown in Figure 2.1 (d), such a recursively prunable convex partition can be represented graphically by a *tree diagram*. For this reason, we use the term “tree” as a shorthand for “recursively prunable convex partition”.

A tree diagram is valuable for interpretability because it breaks even very complex partitions down into simple binary relations. Specifically, it denotes pairs of *sibling* subsets (e.g. x_3 and x_5) that can be pruned to form convex *parent* subsets (e.g. $x_3 \cup x_5$). x_3 and x_5 are said to be the *children* of $x_3 \cup x_5$. Note that x_3 and x_5 have no children themselves, making them *leaves* of the tree. At the top of a tree diagram, there is a single *root* subset with no parent; this corresponds to the full space \mathbb{R}^D . All subsets other than the root and leaves are called *internal*. In addition to bottom-up pruning yielding valid convex partitions of \mathbb{R}^D , the branch of the tree below any internal subset is itself a valid convex partition of that subset. It is thus possible to carry out local analysis of regions of the feature space, independently of the rest of the tree structure. This is especially beneficial for large trees, which may otherwise overwhelm a human analyst if viewed globally. Since the dual diagrammatic and space partitioning representations of trees are complementary, we move fluidly between them throughout this thesis.

This definition of trees in terms of their ability to be pruned is bottom-up, in that it starts by considering the most fine-grained abstraction \mathcal{X} . An efficient way to generate a tree is to reason in the opposite direction, following a top-down algorithm called *binary space partitioning*: first add a single hyperplane to split the feature space into two half-spaces, then recursively split each half-space by adding hyperplanes in the same manner. At every step during the execution of this top-down algorithm, the intermediate result is a valid convex partition of \mathbb{R}^D . The algorithm thus implicitly generates a nested collection of abstractions at varying levels of granularity, not just a single final result. From this perspective, the act of constraining abstractions to be recursively prunable actually *increases* their expressive potential, since it unlocks the additional axis of variable granularity.

2.3.4 Axis-alignment

Recall that our motivation for discussing abstraction is as a mechanism for grounding symbolic expressions in high-dimensional data spaces. In the case of a hierarchical convex partition of a feature space, the geometry of the partition itself can be given a symbolic representation: a collection of hyperplane equations organised into a tree structure. Each equation is associated with the parent of the two subsets it creates (e.g. the first hyperplane partitioning the entire space is located at the root). Given a slightly different interpretation, these hyperplane equations can be viewed as *rules* over linear combinations of features, “ $b_1\mathbf{f}_1 + b_2\mathbf{f}_2 + \dots + b_D\mathbf{f}_D \geq c$ ”,

which determine how feature vectors in \mathbb{R}^D are grouped into the different branches and leaves of the tree. Whether a rule evaluates to true or false for a given $\mathbf{f} \in \mathbb{R}^D$ determines which of the two possible branches is taken from that point onwards. Furthermore, a unique symbolic label can be constructed for every subset in the tree by taking the conjunction of the rules at all of its ancestors. In this way, we obtain a very explicit class of grounded symbols.

However, in the general case, each rule has $D + 1$ parameters (including the threshold c), and we must also keep track of the location of that rule in the tree, meaning a tree with m leaves requires $(D + 1 + 1)(m - 1)$ parameters to specify. This number offers no improvement over Voronoi tessellation in that it is linear in both D and m , with the parameter count being larger whenever $m > 1 + D/2$. This parameter-counting perspective provides one justification for our final geometric desideratum: constraining all partitioning hyperplanes to be *axis-aligned*.

In an axis-aligned tree, one of the coefficients b_1, \dots, b_D in each hyperplane equation/rule is equal to one, and the rest are equal to zero. Letting d denote the index of the nonzero coefficient, this greatly simplifies the form of the rule to “ $\mathbf{f}_d \geq c$ ”, which tests whether the single *split feature* d exceeds a *split threshold* c . As exemplified by Figure 2.1 (e), this simplification makes it possible to display rules directly on the tree diagram. Each rule is now associated with just three parameters: the split feature, threshold and location in the tree. Therefore, a tree with m leaves is fully specified by $3(m - 1)$ parameters, a count which is strictly smaller than Voronoi tessellation for $D \geq 3$. It is also independent of D , making the interpretability of axis-aligned trees especially insensitive to the dimensionality of the underlying domain. Furthermore, the symbolic label for each subset (via ancestor rule conjunction) can be expressed as a conjunction of intervals with a maximum of D terms, regardless of its depth in the tree. In Figure 2.1 (e), the label for x_2 is “ $\mathbf{f}_1 < 0.3$ and $\mathbf{f}_2 \geq 0.4$ ” (the rule “ $\mathbf{f}_1 < 0.7$ ” is subsumed by a tighter upper bound deeper in the tree, so can be dropped), and the label for x_3 is “ $0.3 \leq \mathbf{f}_1 < 0.7$ and $\mathbf{f}_2 \geq 0.4$ ” (the two rules for \mathbf{f}_1 can be grouped into a single interval term).

Parameter counts and symbol lengths are rough proxies for human interpretability, but we can also justify axis-alignment geometrically. Subsets of \mathbb{R}^D bounded by axis-aligned hyperplanes are known as (axis-aligned) *hyperrectangles*. It is plausible that these are considerably easier to reason about in high dimensions than general polytopes, both individually and collectively, since their properties generalise straightforwardly from $D = 2$ and $D = 3$ cases encountered in everyday life (e.g. brick walls, stacked boxes). At all depths in the tree, sibling subsets lie adjacent to each other along one axis in the feature space, and it is easy to conceptualise a single feature crossing a threshold to move a point from one subset to another.

The axis-alignment constraint also brings computational benefits. Throughout this thesis, we exploit the fact that efficient, well understood algorithms exist for generating and analysing axis-aligned trees. In prior work on conceptual spaces, hyperrectangles have been used to enable computationally efficient concept combinations across domains, but these do not form a partitioning or hierarchical structure [21].

Feature Grounding Caveat: The assertion that axis-aligned trees, and their resultant symbolic descriptions, are human-interpretable rests on a foundational assumption that the features comprising \mathbb{R}^D are themselves grounded in natural linguistic quantities (e.g. “height”, “weight”, “speed”, “rotation”, “distance”) or simple combinations thereof. Only then can structures present in a tree be integrated with a human’s prior symbolic knowledge about the conceptual space. This is a kind of ‘garbage in, garbage out’ phenomenon; the basic potential for interpretability is merely preserved, rather than created, by representational constraints such as axis-alignment.

To recap, the preceding subsections started with a general notion of abstraction in conceptual spaces and progressively added constraints to reflect the four geometric desiderata of convexity, partitioning, hierarchy and axis-alignment. While doing so leads to a *prima facie* reduction in expressive flexibility, it increases the likelihood that the resultant abstractions can be understood and reasoned about by a human due to the reduced parameter count and geometric simplicity. It also unlocks the axis of variable granularity, enables additional visual and textual representations in the form of tree diagrams and symbolic rules, and enables computationally efficient algorithms for abstraction generation and analysis.

2.4 Query-Efficient Abstractions

The previous section motivates tree abstractions as a tool for bridging the gap between continuous and symbolic representations of a system, but says little about the ultimate purpose of abstraction, which in our case is to generate human-interpretable summaries of observations of that system. In Section 2.2, we noted that given the inherent loss of information induced by abstraction, good abstractions are those that preserve “*only the information deemed most important for a given downstream application*”. To formalise this notion, imagine posing a set of *queries* about the system, concerning trends and relationships in our observations of it. Given an abstraction, we can compress the continuous conceptual space representation into *summary statistics* at the level of abstract subsets. We would like these summary statistics to answer our queries with similar accuracy to an exhaustive analysis of the data.

Revisiting the language of conceptual spaces, we adopt the following definition:

An abstraction of one domain of a conceptual space is *efficient* to the extent that

- (1) it preserves our ability to answer queries about the other domains via summary statistics,
- while (2) having a small number of parameters.³

Hence, for a conceptual space in $(\mathbb{R}^D, \mathbb{Y})$ form, an efficient tree abstraction of the feature space \mathbb{R}^D (1) captures a maximal amount of query-relevant information about the attributes \mathbb{Y} in its

³This query-centric definition of abstraction efficiency is partly inspired by Millidge [178], although we present a somewhat looser formulation to capture all query types considered in this thesis.

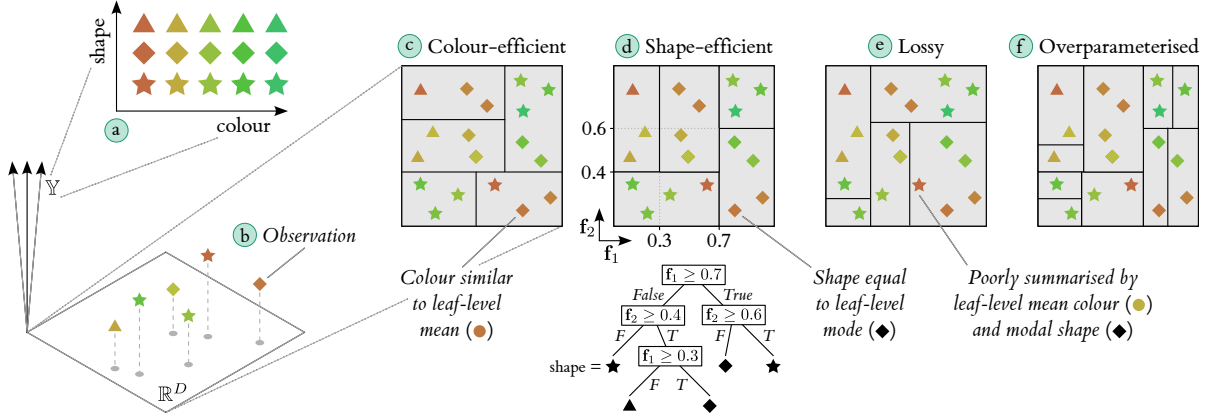


Figure 2.2: Query-centric abstraction efficiency in $(\mathbb{R}^D, \mathbb{Y})$ form conceptual spaces.

summary statistics, while (2) having a small number of *leaves*. Focusing on different queries and evaluation metrics leads to many valid notions of efficiency suitable for different applications.

Figure 2.2 (a) shows an $(\mathbb{R}^D, \mathbb{Y})$ conceptual space with two features f_1, f_2 and two attributes $\mathbb{Y} = \text{colour} \times \text{shape}$. A single observation (b) is thus a 2D feature vector paired with a coloured shape. Given a dataset of many such observations, we may want to understand how either the colour or shape attribute varies across the feature space. In the absence of abstraction, the only available description of the data is a complete one, i.e. all individual feature vectors and their coloured shapes. All analytical workload is left to a human interpreter, which presents an increasing cognitive challenge as the number of observations, or the dimensionality D of the feature space, grows.

Now consider introducing an abstraction of \mathbb{R}^D , such as the trees in Figure 2.2 (c) - (f). We can summarise the attribute information in the dataset by grouping observations by the leaves that contain them, and performing a statistical averaging operation (e.g. mean for continuous attributes, mode for discrete attributes) at the leaf level. We can then discard the raw observations and attempt to answer queries using the much smaller set of leaf-level summary statistics. The efficiency of the abstraction with respect to a given query depends on how representative these summary statistics are of the raw data. In turn, this depends on how judiciously observations are grouped into leaves.

For example, Figure 2.2 (c) is an efficient abstraction with respect to queries about colour values. Since the observations in each leaf all have similar colours, they are well summarised by their leaf-level mean. Meanwhile, (d) is efficient for shape queries because the observations in each leaf have a common shape. In fact, (d) is a *lossless* abstraction for shape because the shape of each observation is identical to its leaf-level mode. The tree diagram for abstraction (d) is plotted below it, with the unique shape associated with each leaf shown. This diagram provides a compact symbolic summary of all feature-shape dependencies in the dataset.

Abstractions can be inefficient for two reasons, exemplified by Figure 2.2 (e) and (f). The leaves of (e) have high internal variability in both colour and shape, so an abstraction-based

summary is a poor representation of the data for both query types. Meanwhile, (f) is a fine-graining of (d), obtained by adding five more axis-aligned hyperplane rules. It is inefficient because it adds needless parameters to an abstraction that was already lossless for shape queries. An effective abstraction algorithm would thus return the tree in the state shown in (d), rather than including the unnecessary rules in (f). As alluded to above, the inherent hierarchy of tree abstractions allows us to use pruning to control the tradeoff between summary accuracy and parameter count for an efficient compromise. Although there is rarely a single optimal point on this tradeoff, pruning should yield a *Pareto efficient* tree from which further reducing parameter count reduces accuracy, and further increasing accuracy increases parameter count.

Crucially, there is no reason in general to expect an efficient abstraction for one type of query to be efficient for another. In Figure 2.2, (c) is not particularly shape-efficient, and (d) is not particularly colour-efficient. This underlines the fundamental query-dependence of any notion of efficiency, and the need for specialised abstractions.

In the colour and shape examples above, we have assumed simple queries are made about the value of one attribute. In such cases, the accuracy of a tree-based summary depends on a measure of spread in that attribute over the observations contained in each leaf, typically entropy or Gini impurity for a discrete-valued attribute (e.g. shape in Figure 2.2), or variance for a continuous-valued attribute (e.g. colour) [32]. Optimising trees for such single-attribute queries corresponds to the canonical problems of classification and regression in supervised machine learning, which are by far the most common applications of tree abstractions. Although the sequence of tree-based methods in this thesis begins with a minor departure from standard classification and regression tree (CART) algorithms, we find that a holistic treatment of the interpretability problem in the context of dynamic agents and environments involves multi-attribute queries, and those that are *non-local* in that they depend on interactions between leaves rather than independent variability measures. This motivates novel tree abstraction methods optimised for more sophisticated queries, collectively forming a toolbox of abstract representations that serve different downstream applications.

2.5 Abstractions of Agents and Environments

After a departure from the terminology of the previous chapter, we are now ready to integrate our ideas about interpretable and efficient abstractions back into the core narrative of this thesis: building summaries of agent behaviour and learning in dynamic environments.

In [95], Gärdenfors and Warglien propose an extension of conceptual spaces theory to actions and events, with a view to providing semantic grounding for verbs in natural language. This so-called *two-vector* model involves a domain separation into a space of *states* of some system $s \in \mathcal{S}$ and a space of *actions* $a \in \mathcal{A}$ generated by an agent.⁴ The two domains are

⁴The original notation uses P for the state space and A for the action space.

related by a mapping from actions to *forces*, which change the state $s \rightarrow s'$ and produce *result vectors* $(s, s') \in \mathcal{S} \times \mathcal{S}$. A sequence of action-induced state changes forms a *trajectory* in state space. Gärdenfors and Warglien define abstract generalisations of states as convex subsets of \mathcal{S} , abstract actions as convex subsets of \mathcal{A} , and *events* as convex subsets of the three-way product space $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ (which contains actions and result vectors). Let this product space be denoted the *transition space*. Simple verb-noun pairings can be grounded in events in a transition space. For example, “moving a book” is grounded in a set of actions (such as robot motor commands) that produce a nonzero result vector in positional dimensions of the the book’s state space. Gärdenfors and Warglien note that the conceptual space can be furnished with other dimensions and domains; for consistency with the previous section, let us refer to these collectively as attributes \mathbb{Y} . Among the attributes, they postulate a *reward* dimension to account for the motivations of intentional agents.

As the reader may have noticed, this model strongly resembles a Markov decision process, as formalised in Section 1.2.⁵ We thus have reason to believe that the interaction between an agent and its environment can be understood as a conceptual space. Figure 2.3 (a) depicts a single observation in such a space, which consists of an agent action $a \in \mathcal{A}$, an environment state result vector $(s, s') \in \mathcal{S} \times \mathcal{S}$, and values for whichever other attributes \mathbb{Y} are being measured. As shown in (b), this may include a real-valued reward (red-blue colour scale).

Agent-environment interaction can be a complicated dynamical process, with dependencies across its various conceptual space domains, and over time as sequences of observations are made. To enable human interpretation of agent behaviour and learning, some mechanism for simplifying and summarising these observations is essential. Throughout this thesis, we do this by building tree abstractions. Rather than talking about the system on an observation-by-observation basis, we may then analyse the leaves of a tree as meaningful entities in themselves, within which the agent behaves in predictable ways, and between which it moves in predictable patterns. First, however, we must wrangle the conceptual space in Figure 2.3 (a) into the $(\mathbb{R}^D, \mathbb{Y})$ form assumed in Sections 2.3 and 2.4. This is done by a process that we call *featurisation*:

1. Identify one domain to be the target of abstraction, $dom \in \{\mathcal{S}, \mathcal{A}, \mathcal{S} \times \mathcal{A} \times \mathcal{S}\}$.
2. Define a feature function $\phi : dom \rightarrow \mathbb{R}^D$ that maps points in dom to real vectors.
3. Bundle all other domains into the attributes \mathbb{Y} .

Revisiting the Feature Grounding Caveat: Featurisation is the step at which the basic potential for interpretability is created or destroyed. For an abstraction of an agent-environment system to be understood by a human, the features produced by ϕ must be aligned with natural linguistic quantities or simple combinations thereof. In practice, ϕ may not strictly be required:

⁵The absence of references to this connection suggests that Gärdenfors and Warglien are unaware of it, and we have yet to find any citing articles that mention it.

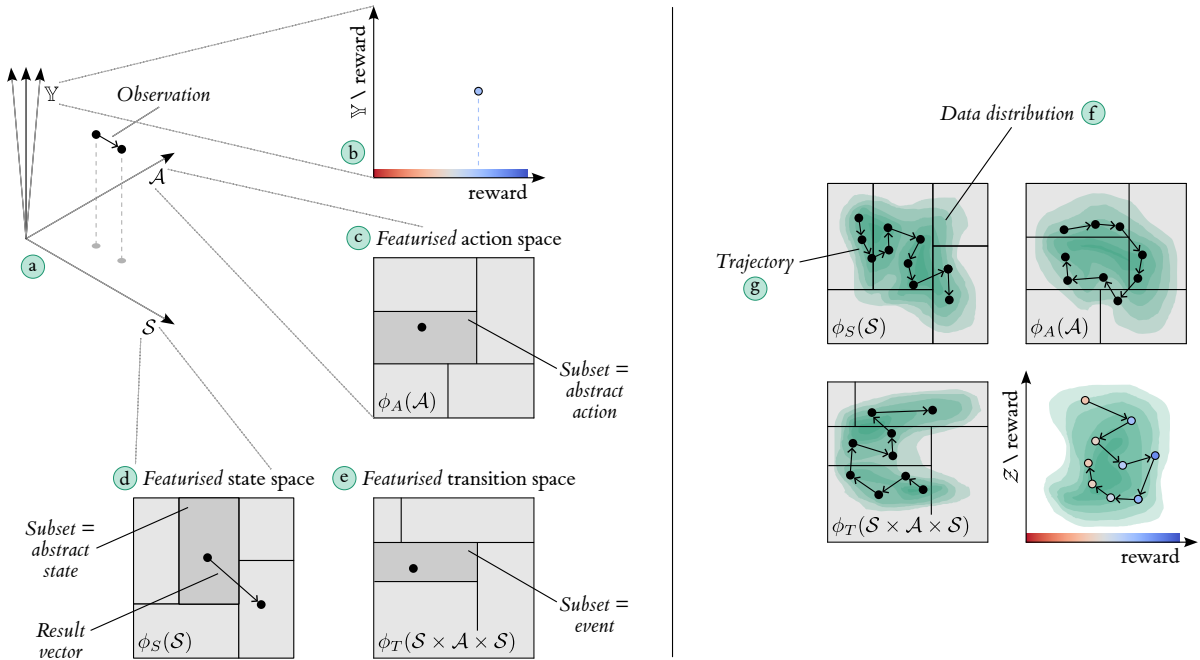


Figure 2.3: A conceptual spaces model of agent-environment interaction.

benchmark environments for learning agents often expose states and actions as real vectors by default as this is a convenient form for neural network-based learning architectures (e.g. the `spaces.Box` class in Gymnasium [86]). However, even in these cases, there may be value in transforming the features into a more naturally interpretable form (e.g. moving between trigonometric and direct representations of robot joint angles). Appropriate featurisation is important for all methods presented in this thesis. In the next chapter, we explore one possible way to programmatically generate a large bank of interpretable candidate features, which are then downselected by a tree induction algorithm.

Figure 2.3 (c), (d) and (e) show a featurised state space $\phi_S(\mathcal{S})$, action space $\phi_A(\mathcal{A})$ and transition space $\phi_T(\mathcal{S} \times \mathcal{A} \times \mathcal{S})$, with a single observation. Slightly refining Gärdenfors and Warglien’s terms, let us define abstract states as convex subsets of $\phi_S(\mathcal{S})$, abstract actions as convex subsets of $\phi_A(\mathcal{A})$, and events as convex subsets of $\phi_T(\mathcal{S} \times \mathcal{A} \times \mathcal{S})$. Illustrative tree abstractions of all three spaces are shown.

The efficiency of an abstraction can only be evaluated relative to a particular distribution of observational data, an example of which is illustrated by the green shading (f) in the plots on the right of Figure 2.3. In the case of a learning agent, this distribution changes over time across all domains. Also shown is the projection of a single trajectory (g) into the various (featurised) domains. It is with reference to potentially changing data distributions and temporally extended trajectories, not merely a set of independent observations, that a holistic understanding of learning agent behaviour is gained. For this reason, several methods presented in this thesis explicitly use temporal information in their definitions of abstraction efficiency.

Continuing the narrative from Section 2.4, abstraction efficiency also depends on the particular set of queries we pose about the agent and our observations of it. For example, efficient abstract states might be those containing observations with a common action, thus enabling accurate query-answering about actions taken in different parts of the state space (“in this circumstance, what does the agent do?”). Conversely, efficient abstract actions might be those correlated with a consistent change of value along one or more dimensions of the state space, enabling the answering of queries in the opposite direction (“what is the effect when the agent does this?”). Events might be efficient if they contain a narrow distribution of reward values, enabling a summary of the normative information in the system (“when this happens, how good is it?”). Each type of query only provides partial insight. This implies the need for a toolbox of abstraction-based representations, as developed in this thesis.

In summary, the important takeaways from this section are as follows:

- The state space \mathcal{S} , action space \mathcal{A} and other attributes \mathbb{Y} of an agent’s interaction with its environment (e.g. rewards) define a conceptual space.
- A (learning) agent’s interaction history induces a (changing) data distribution in the conceptual space. A dataset of observations of that history is an ordered sequence of state-action-next-state tuples (s, a, s') with associated attribute values.
- An abstract representation of these data can be obtained by wrangling the conceptual space into $(\mathbb{R}^D, \mathbb{Y})$ form via featurisation, defining a tree abstraction of \mathbb{R}^D , and computing summary statistics over \mathbb{Y} for each leaf.
- The efficiency of this abstraction depends on the set of queries we pose about the system, which in turn depends on the downstream application. Different queries necessitate different abstractions.

2.6 Conclusion

This chapter has taken a deep dive into a theory of symbol grounding through abstraction, and a query-centric definition of abstraction efficiency, before emerging to connect these ideas back to our objective of understanding agent behaviour and learning. Although the remainder of this thesis would have been coherent without this conceptual background, we consider it beneficial to establish a common grounding.

This view can be justified by existing trends in AI interpretability. While the field has been historically fragmented, with an ad hoc mixture of methods and evaluation metrics, some of the most influential works propose generalisable algorithm schemas founded on sound principles and desiderata. For example, *local interpretable model-agnostic explanation* (LIME) is a general technique for understanding the local shape of a predictive function by fitting an interpretable surrogate model to perturbations of a reference input [208]. The architecture of the surrogate,

and the mode of perturbation, can be tailored to the context. The intuitive and theoretical basis of LIME is shared by its many derivative variants, allowing their position in the interpretability landscape to be easily identified. In the agent context, interpretation via *critical examples* is another powerful general schema. Given some measure of the task-specific importance of a given state (often derived from the agent’s policy or value function), the aim here is to assemble a small but diverse set of states [122] or trajectories [6] that summarise the agent’s global strategy. This approach could be specialised in many ways for a desired application.

Our aim in this chapter has been to present query-efficient abstraction with trees as another flexible algorithm schema, whose interpretability credentials can be traced back to cognitive science. The schema unifies the methods in this thesis with a common mathematical language, enabling their synergy into a cohesive toolkit, and provides a context in which methods specialised for other query types could be developed in future.

Trees are seen as a gold standard of machine learning interpretability [180, 219], and have been employed in various ways to interpret agents (see Section 1.3), but the justification for their use over other model architectures is often absent or vague. To our knowledge, the only prior attempts to seriously investigate the foundations of tree interpretability are [235] and [205]. Like us, the author highlights trees’ dual interpretation as both geometric objects consisting of simple hyperrectangular regions and symbolic objects consisting of hierarchies of rules. Also mentioned are the ability to describe prediction by a sequence of easy-to-follow binary decisions, the automatic feature selection effect of axis-alignment, and the potential for local analysis of tree branches. By discussing these properties within the broader context of the symbol grounding problem and conceptual spaces theory, we have shown that trees can be understood as a solution to a set of desiderata for human-interpretable representations of multidimensional data. We believe this has implications outside the agent context.

In the remaining chapters of this thesis, we build on the theory and notation established in this chapter to develop a diverse set of tree-based models for agent interpretability.

Chapter 3

Tree Models of Agent Behaviour

Based on: “Modelling Agent Policies with Interpretable Imitation Learning”, published in Trustworthy AI - Integrating Learning, Optimization and Reasoning, Springer LNCS, 2021; and “TripleTree: A Versatile Interpretable Representation of Black Box Agents and their Environments”, published at 2021 AAAI Conference on Artificial Intelligence.

3.1 Introduction

In this chapter, we use the language of tree abstractions to construct interpretable models of an agent executing a stationary (i.e. unchanging) policy in its environment, and explore how these models provide understanding through prediction, visualisation and rule-based explanation. No assumption is made about the agent’s provenance; its policy may be a product of RL, optimal control algorithms, evolution, or explicit manual design. To learn anything about the agent we must collect data, and in doing so we take a *black box* perspective [51, 102, 261]. That is, we adopt the role of a passive spectator of the agent-environment complex, with no access to the internal structure of either system, but with the ability to directly observe environment states, agent actions, and instantaneous rewards (all of which are assumed to be externally visible), and their order of occurrence over time.

Given data collected under these conditions, we present two tree-based models, optimised for different kinds of query about the agent’s behaviour. The first focuses exclusively on the relationship between states and actions, and learns a tree abstraction of the state space that enables accurate reconstruction of the agent’s policy. After discussing the limitations of such a policy-only model, we then extend the approach to also capture invariances in the agent’s value function (expected future reward) and dynamics (state changes between timesteps). In the process of developing these models, we propose a framework for synthesising an interpretable feature space for use in tree induction, examine the tree performance/complexity tradeoff through pruning, present algorithms for counterfactual explanation of tree-based predictions and how they change over time, and develop a novel method for visualising a tree’s induced partition in high-dimensional feature spaces.

3.2 Tree Models of Black Box Policies

We first consider how trees can be constructed to summarise a black box agent’s policy. This general strategy is perhaps the one that most naturally springs to mind when the problem of agent interpretability is first encountered, since it is a direct analogue of *surrogate modelling* for supervised learning [180], and at the time this work was completed, it already had some precedence in the literature [20, 51, 151] (see Section 3.6.1 for more related work). Going beyond these prior methods, we propose a new way of synthesising a large bank of interpretable features from the state space by recursive application of a set of elementary operators, which are then downselected by the tree induction algorithm, and provide a detailed examination of how a tree model can be used to generate rule-based explanations of the policy’s actions and their changes over time. Primarily, however, this preliminary study is best viewed as establishing a point of departure for the sequence of methods presented later in this chapter and thesis, which find more novel roles for tree abstractions in the effort to render artificial agents more understandable to humans.

3.2.1 Generic Problem Statement

Let us first describe the problem of interpretable policy modelling in generic terms. We adopt the perspective of a passive spectator of a black box agent executing a stationary policy in its environment, with the ability to observe the environment’s state and the action taken by the agent at each time step. This allows us to record an interaction history of N states and actions $\mathcal{D} = \{(s_1, a_1), \dots, (s_N, a_N)\}$ to use as the basis of model learning. Our objective is to identify the underlying mapping between states and actions given \mathcal{D} . Without loss of generality, this mapping can be decomposed into a state-to-feature function $\phi : \mathcal{S} \rightarrow \mathbb{R}^D$ and a feature-to-action policy $\pi' : \mathbb{R}^D \rightarrow \Delta(\mathcal{A})$.¹ The decomposition is not restrictive in itself because it makes no assumptions about ϕ : if $\pi'(\phi(s)) = \phi(s)$ for example, it is possible to reconstruct any policy exactly. However, we also wish to constrain the search spaces for ϕ and π' (Φ and Π respectively) so that they only contain functions that are human-interpretable. This property must be achieved while minimally sacrificing reconstruction accuracy or tractability of the modelling problem. The objective can be formulated as an optimisation:

$$(3.1) \quad \operatorname{argmin}_{\phi \in \Phi, \pi' \in \Pi} \left[\mathbb{E}_{(s,a) \in \mathcal{D}} \mathbb{E}_{a' \sim \pi'(\cdot | \phi(s))} [\ell(a, a')] \right], \text{ where } \Phi, \Pi = \text{“interpretable”},$$

and $\ell : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}_{\geq 0}$ is a pairwise loss function over the environment’s action space. In other words, we want to reconstruct the state-action mapping as accurately as possible, subject to the interpretability constraints. The schematic in Figure 3.1 outlines the task at hand.² We now present concrete proposals for a Φ and Π that meet the interpretability requirement.

¹We use π' to denote the learnt policy model; π is the true policy, which we cannot observe directly.

²Icons from users *Freepik* and *Pixel Perfect* at www.flaticon.com.

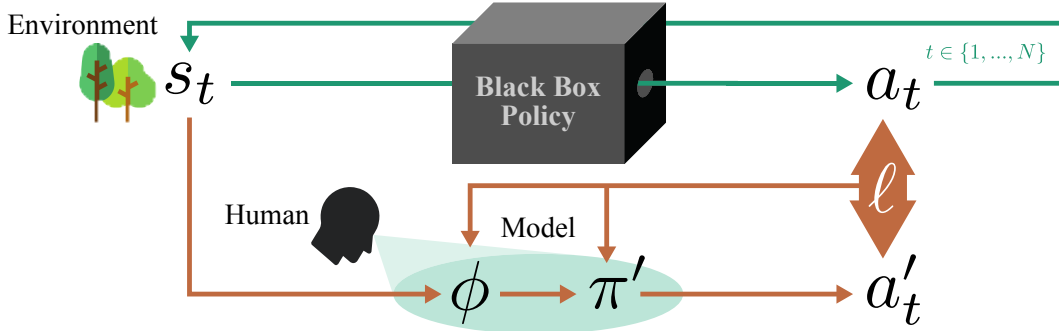


Figure 3.1: Generic problem setup for interpretable policy modelling. The objective is to minimise the loss between a_t and $a'_t \forall t \in \{1, \dots, N\}$ while ensuring both the feature function ϕ and policy model π' are comprehensible under human scrutiny.

3.2.2 Recursive Feature Generation

Φ should permit only human-interpretable features representing the environment state. As per the **Feature Grounding Caveat** (Section 2.3.4), the availability of an interpretable feature space is crucial for the resultant interpretability of any model built on top. This subsection describes a flexible framework for generating a large number of such features automatically.

We consider features that can be synthesised from the underlying environment state s by recursively applying a finite set of elementary *operators* \mathbb{O} , each of which has a well-defined domain and codomain and is taken to be functionally interpretable in itself. Generic examples might include basic numerical operators such as addition and subtraction, and Boolean operators such as negation, conjunction and disjunction. Other, more domain-specific, operators may also be relevant in some cases. In the traffic simulator experiments described later, \mathbb{O} contains operators to extract a vehicle’s speed and position, find the nearest vehicle and junction ahead of and behind a position, and subtract one speed or position from another. As will be shown later, a large and diverse set of features can be generated by recursively combining these simple primitive operators.

Formally, we say that an operator $o \in \mathbb{O}$ is *valid* for given input if the range of possible values for that input is a subset of the domain of the operator (e.g. the interval $[0, 1]$ is a subset of \mathbb{R} , as is \mathbb{R} itself). A feature is valid, and thus constructible by the recursive algorithm, if it comprises entirely of valid operations. To limit the complexity of Φ , we also specify a limit r on the depth of recursion. As an illustrative example, suppose \mathbb{O} contains three operators:

$$o_1 : \mathcal{S} \rightarrow [0, 1], \quad o_2 : \mathcal{S} \rightarrow [-1, 1], \quad \text{and} \quad o_3 : \mathbb{R} \times \mathbb{R} \rightarrow \{1, 2, 3\},$$

and the recursion depth limit is set at $r = 2$. Here,

$$f_1(s) = o_1(s) \quad \text{and} \quad f_2 = o_3(o_2(s), o_1(s))$$

are valid features of the state s , but

$$o_3(s) \quad \text{and} \quad o_1(o_2(s))$$

are both invalid operator combinations due to input-domain mismatches, and while valid,

$$o_3(o_3(o_1(s), o_2(s)), o_1(s))$$

is not generated because it exceeds the depth limit (it has a depth of 3). Consequently,

$$\phi(s) = [f_1(s), f_2(s)]$$

is a vector of valid features with dimensionality $D = 2$, making ϕ a member of the set of possible feature functions Φ .

Note the core assumption that lies at the heart of this approach: given a set of canonical operators whose interpretability is taken as self-evident, valid combinations of those operators (up to a limited depth) retain that interpretability by virtue of their traceable derivation. With this formulation, prior domain knowledge is used to design \mathbb{O} , which in turn has a shaping effect on ϕ without requiring the stronger assumption of which precise features to use. We suggest that this could be a general and scalable route to feature design for interpretable learning algorithms beyond the realm of policy modelling.

3.2.3 Tree-Structured Policy Model

In further service of the goal of interpretability, the policy space Π should be limited to functions that are comprehensible to humans while retaining sufficient representational capacity for high reconstruction fidelity (predictive accuracy with respect to the target policy). In this work, as throughout this thesis, we achieve this by enforcing a tree structure.

Following the notation introduced in Chapter 2, let $\mathcal{X} = \{x_1, x_2, \dots, x_m\}$ denote the leaves of a tree over a D -dimensional feature space $\phi(\mathcal{S}) = \mathbb{R}^D$, which in this case has been constructed by the recursive operator method described above. For each element (s, a) in the dataset of state-action observations \mathcal{D} , the state feature vector $\phi(s)$ lies within exactly one leaf. We can thus define \mathcal{D}_x as the subset of observations that belong to leaf x :

$$(3.2) \quad \mathcal{D}_x = \{(s, a) \in \mathcal{D} : \phi(s) \in x\}.$$

Recall that in Section 2.4 we discussed how a tree abstraction allows us to build a compressed representation of a dataset by computing summary statistics at the leaf level. For the present purposes, we wish to summarise the distribution of actions taken by the agent in each leaf. When the action space \mathcal{A} is discrete, this categorical distribution can be represented in full:

$$(3.3) \quad \Pr(a|x) = \frac{|\{(s, a') \in \mathcal{D}_x : a' = a\}|}{|\mathcal{D}_x|}, \quad \forall a \in \mathcal{A}.$$

When \mathcal{A} is continuous, a sensible choice of parametric summary is the normal distribution:³

$$(3.4) \quad \Pr(a|x) = \mathcal{N}(a|\mu_x, \sigma_x^2), \quad \text{where} \quad \mu_x = \mathbb{E}_{(s,a) \in \mathcal{D}_x} [a] \quad \text{and} \quad \sigma_x^2 = \mathbb{E}_{(s,a) \in \mathcal{D}_x} [(a - \mu_x)^2].$$

³Strictly speaking, this is a density function, but we use the same notation as for discrete actions for simplicity.

In either case, the leaf-level statistical summaries can be used to define a predictive function that outputs a distribution over actions given an arbitrary state $s \in \mathcal{S}$ as input. This function is what we take to be π' , the tree-based model of the agent's policy π :

$$(3.5) \quad \pi'(a|\phi(s)) = \Pr(a|x), \quad \text{where } \phi(s) \in x.$$

Alternatively, a deterministic policy can be constructed by taking the modal (for discrete \mathcal{A}) or mean (for continuous \mathcal{A}) value of each leaf-level distribution. For simplicity, we use a deterministic form when constructing textual policy explanations in Section 3.5.

We now introduce another statistic that will prove crucial for the development of algorithms for efficient tree induction. Let the action *impurity* of a leaf x be the expected action loss between pairs of observations drawn uniform-randomly from its data subset:

$$(3.6) \quad \text{I}^A(\mathcal{D}_x) = \mathbb{E}_{(s,a) \in \mathcal{D}_x} \mathbb{E}_{(s',a') \in \mathcal{D}_x} [\ell(a, a')],$$

where ℓ is the pairwise action loss function introduced in Section 3.2.1. Note that for discrete action spaces, the popular Gini impurity measure is recovered by defining $\ell(a, a') = 0$ if $a = a'$, and $\ell(a, a') = 1$ otherwise. For continuous actions, leaf impurity is commonly quantified in terms of variance. This equates (up to a scale factor of $\frac{1}{2}$) to defining $\ell(a, a') = (a - a')^2$.⁴

The policy modelling objective in Equation 3.1 can be expressed in terms of leaf impurity. To demonstrate this, let us make use of Equations 3.2 and 3.5 to rewrite the objective's minimand:

$$(3.7) \quad \mathbb{E}_{(s,a) \in \mathcal{D}} \mathbb{E}_{a' \sim \pi'(\cdot|\phi(s))} [\ell(a, a')] = \sum_{x \in \mathcal{X}} \frac{|\mathcal{D}_x|}{|\mathcal{D}|} \mathbb{E}_{(s,a) \in \mathcal{D}_x} \mathbb{E}_{a' \sim \Pr(\cdot|x)} [\ell(a, a')],$$

From this rewriting, it is clear that reconstruction fidelity depends on the expected deviation between the actions of observations at each leaf x and samples from the summary distribution $\Pr(a|x)$, with greater weighting given to leaves containing more observations. Notice how this provides our first concrete example of the notion of data being *well summarised* by leaf-level statistics, which is introduced in an abstract sense in Section 2.4. Also notice the structural similarity to the impurity definition in Equation 3.6, insofar as a loss expectation is taken over two sampled actions a and a' . If we can show that this expectation is the same regardless of whether a' is sampled from the data subset \mathcal{D}_x or from the summary distribution $\Pr(a|x)$, then it would be permissible to further rewrite the minimand as follows:

$$(3.8) \quad \mathbb{E}_{(s,a) \in \mathcal{D}} \mathbb{E}_{a' \sim \pi'(\cdot|\phi(s))} [\ell(a, a')] = \sum_{x \in \mathcal{X}} \frac{|\mathcal{D}_x|}{|\mathcal{D}|} \cdot \text{I}^A(\mathcal{D}_x).$$

Indeed, the required condition holds for both discrete actions with the Gini impurity measure,

$$\begin{aligned} \mathbb{E}_{(s',a') \in \mathcal{D}_x} [\ell(a, a')] &= \frac{1}{|\mathcal{D}_x|} \sum_{(s',a') \in \mathcal{D}_x} [a' \neq a] = \frac{|\mathcal{D}_x| - |\{(s',a') \in \mathcal{D}_x : a' = a\}|}{|\mathcal{D}_x|} \\ &= 1 - \Pr(a|x) = 1 - \mathbb{E}_{a' \sim \Pr(\cdot|x)} [a' = a] = \mathbb{E}_{a' \sim \Pr(\cdot|x)} [a' \neq a] = \mathbb{E}_{a' \sim \Pr(\cdot|x)} [\ell(a, a')], \end{aligned}$$

⁴This can be generalised to multidimensional continuous action spaces as the (weighted) Euclidean distance.

and for continuous actions with normal summaries and the variance impurity measure,

$$\begin{aligned}
 & \mathbb{E}_{(s,a) \in \mathcal{D}_x} \mathbb{E}_{(s',a') \in \mathcal{D}_x} [\ell(a, a')] = \mathbb{E}_{(s,a) \in \mathcal{D}_x} \mathbb{E}_{(s',a') \in \mathcal{D}_x} [(a - a' + \mu_x - \mu_x)^2] \\
 = & \mathbb{E}_{(s,a) \in \mathcal{D}_x} [(a - \mu_x)^2] + \mathbb{E}_{(s',a') \in \mathcal{D}_x} [(a' - \mu_x)^2] - 2 \mathbb{E}_{(s,a) \in \mathcal{D}_x} [(a - \mu_x) \mathbb{E}_{(s',a') \in \mathcal{D}_x} [(a' - \mu_x)]] \quad = 0 \\
 = & 2 \mathbb{E}_{(s,a) \in \mathcal{D}_x} [(a - \mu_x)^2] + 0 = 2 \mathbb{E}_{(s,a) \in \mathcal{D}_x} [(a - \mathbb{E}_{a' \sim \text{Pr}(\cdot|x)} [a'])^2] = \mathbb{E}_{(s,a) \in \mathcal{D}_x} \mathbb{E}_{a' \sim \text{Pr}(\cdot|x)} [(a - a')^2] \\
 & = \mathbb{E}_{(s,a) \in \mathcal{D}_x} \mathbb{E}_{a' \sim \text{Pr}(\cdot|x)} [\ell(a, a')],
 \end{aligned}$$

where the step highlighted in green exploits a well-known relationship between squared deviation from the mean and squared pairwise differences [274]. The form given in Equation 3.8 is thus valid in both of these standard cases.

This result implies that to achieve good reconstruction fidelity with respect to the target policy, we should aim to construct a tree that minimises the weighted sum of action impurity across its leaves. This insight provides the basis for a tractable learning algorithm.

3.2.4 Modelling Procedure

We combine ideas from the preceding subsection to define the following procedure for optimising the policy modelling objective in Equation 3.1. The procedure is represented in Figure 3.2, using a similar style of visualisation to that used in Chapter 2. Firstly, domain knowledge must be applied to manually define a set of feature-generating operators \mathbb{O} , and specify a recursion depth r . With these factors in place, it is straightforward to enumerate *all* valid features; doing so defines a maximal feature space $\phi_{\text{all}}(\mathcal{S})$. Let D_{all} be the dimensionality of this space. While the tree model is initially constructed in this maximal space, it will later be possible to discard any feature that is not used in any rule in the tree, usually resulting in a final feature space $\phi(\mathcal{S})$ with much-reduced dimensionality $D \ll D_{\text{all}}$ (note that the maximum possible value for D is $m - 1$, the total number of rules in the tree).

The process used for learning the tree itself consists of two stages: *growth* and *pruning*. The growth stage follows the standard CART algorithm [32], which iteratively adds partitioning rules one at a time in a top-down, *greedy* manner, starting from the root of the tree.

By “*greedy*”, we mean that the criterion for selecting between candidate partitions considers only the immediate reduction in loss, rather than the loss of the final (as-yet-unknown) state of the tree. This is suboptimal but regularly performs well in practice, especially when followed by a well-designed pruning stage, such as the one described below. Alternative algorithms exist that *look ahead* two or more partitioning decisions into the future, but these are far more computationally expensive, often for little to no reduction in loss [185]. While the problem of constructing truly optimal trees is NP-complete for most interesting loss functions [152], ongoing improvements in computing hardware mean that this is now possible for moderate

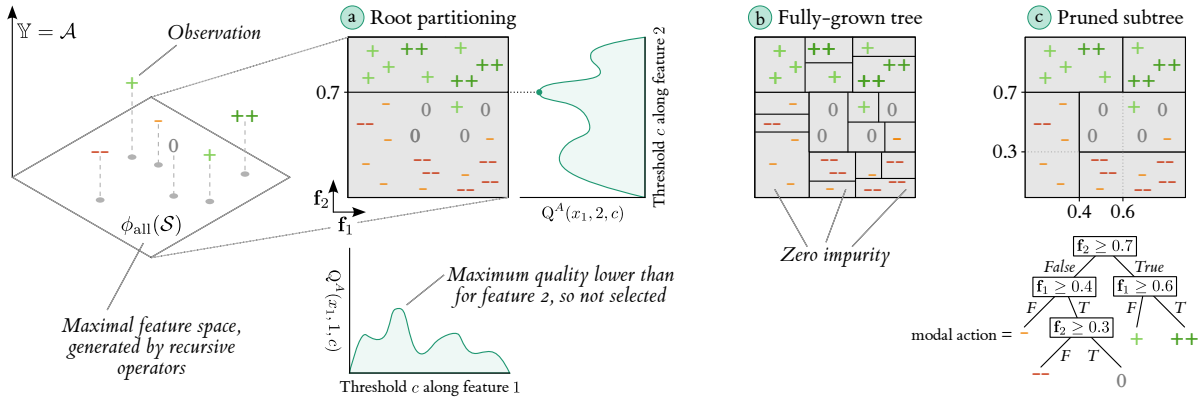


Figure 3.2: A state-action dataset as observations in a conceptual space in $(\phi_{\text{all}}(\mathcal{S}), \mathbb{Y})$ form, where $\phi_{\text{all}}(\mathcal{S})$ is the maximal state feature space and the only attribute is the agent’s action, $\mathbb{Y} = \{\mathcal{A}\}$. Here, \mathcal{A} contains five discrete actions, $\mathcal{A} = (--, -, 0, +, ++)$, which match those in our traffic simulator experiments. Also shown are tree abstractions at three stages during the modelling procedure: splitting the root, growing to zero impurity, and pruning.

datasets via mixed-integer programming [25]. We watch progress in this area with great interest. However, all tree induction algorithms presented in this thesis follow the greedy paradigm. In part, this is because our focus is on exploring new kinds of abstraction and the human-agent understanding that they enable, and additionally tackling questions of optimality would expand the scope beyond a practical level. We also suggest that the simplicity of greedy algorithms makes them functionally interpretable in themselves, providing a valuable opportunity to explain the provenance of the trees that they generate.

Top-down tree growth is a recursive process. At each point during this process, there exists an extant tree with $m \geq 1$ leaves $\mathcal{X} = \{x_1, \dots, x_m\}$, forming a partition of the maximal feature space $\phi_{\text{all}}(\mathcal{S})$. One step of growth involves selecting a leaf $x \in \mathcal{X}$ and adding a rule, or from a geometric perspective, an axis-aligned hyperplane, to split it into two new leaves. Recall from Section 2.3.4 that a rule has the form “ $\mathbf{f}_d \geq c$ ” where $d \in \{1, \dots, D_{\text{all}}\}$ is a split feature and $c \in \mathbb{R}$ is a split threshold. The new leaves that would be created by any given d, c pair are

$$(3.9) \quad x^{(d \geq c)} = \{\mathbf{f} \in x : \mathbf{f}_d \geq c\} \quad \text{and} \quad x^{(d < c)} = x \setminus x^{(d \geq c)}.$$

In the greedy growth paradigm, we search for the d, c pair that minimises the loss of the resultant $(m + 1)$ -leaf tree, given a dataset of state-action observations \mathcal{D} . By Equation 3.8, this is achieved by minimising the weighted sum of action impurity across all leaves. Crucially, however, impurity is a local property of each leaf, so the impurity contribution of all other leaves $\mathcal{X} \setminus \{x\}$ will remain unchanged regardless of which d, c pair is chosen. The *quality* (loss reduction) of a candidate split of x can thus be defined in terms of x , $x^{(d \geq c)}$ and $x^{(d < c)}$ only:

$$(3.10) \quad Q^A(x, d, c) = |\mathcal{D}_{x^{(d \geq c)}}| \cdot I^A(\mathcal{D}_{x^{(d \geq c)}}) + |\mathcal{D}_{x^{(d < c)}}| \cdot I^A(\mathcal{D}_{x^{(d < c)}}) - |\mathcal{D}_x| \cdot I^A(\mathcal{D}_x),$$

where $\mathcal{D}_{x^{(d \geq c)}}$ and $\mathcal{D}_{x^{(d < c)}}$ are the subsets of \mathcal{D} that belong to the two new leaves.

The CART algorithm identifies the greedily-optimal d, c pair through exhaustive search. To make the search over c tractable, we must restrict the options to a finite set of candidate split thresholds $\mathcal{C}_d \subset \mathbb{R}$ along each feature d . It is common to define \mathcal{C}_d as all midpoints between adjacent unique feature values in the dataset, and we follow that precedent in this work. The greedy objective when splitting a leaf x is thus the following:

$$(3.11) \quad \operatorname{argmax}_{1 \leq d \leq D_{\text{all}}, c \in \mathcal{C}_d} \left[Q^A(x, d, c) \right].$$

Top-down tree growth begins with an *empty* tree consisting of a single root. In the geometric language of space partitioning, this equates to a trivial abstraction of $\phi_{\text{all}}(\mathcal{S})$ with a single subset $\mathcal{X} = \{x_1\}$, where $x_1 = \phi_{\text{all}}(\mathcal{S})$. The first step of growth is thus to split the root. Figure 3.2 (a) shows an illustrative example of the search for a greedily-optimal d, c pair for root splitting. The green curves on the two axes below and to the right represent the quality $Q^A(x_1, d, c)$ of each possible threshold c along features $d = 1$ and $d = 2$ respectively. In this case, the single highest quality is attained with $d = 2, c = 0.7$, so this split is chosen.

From this point onwards, the standard CART algorithm proceeds to split leaves recursively in a depth-first manner until a stopping criterion, such as a maximum depth or impurity threshold, is met. In the experiments described later, where the action space is discrete, we let growth continue until every leaf has zero impurity (i.e. all contained observations have the same action). An example of a tree with zero impurity is shown in Figure 3.2 (b).

With zero impurity as the stopping criterion, the fully-grown tree is likely to be very large for reasonably-sized datasets, which greatly hampers its overall interpretability (it is also liable to be overfitted and thus generalise poorly to unseen data, especially when the agent’s policy is stochastic). In the language of Section 2.4, we wish to find a more *efficient* tree that achieves an appropriate compromise between loss and parameter count. At this point, the hierarchical structure of trees becomes extremely beneficial, because it automatically provides a nested collection of abstractions to choose from, at varying levels of granularity. This nested collection is traversed by *pruning* the fully-grown tree, which involves progressively merging its leaves (or in diagrammatic terms, removing its branches) to create smaller subtrees. In Figure 3.2, an example of one possible pruned subtree of (b) is shown in (c). Among the collection of nested possibilities, we aim to find the subtree that is most efficient in the sense described below.

For a tree \mathcal{X} , a branch is a set of leaves that can be merged to form a single convex subset. The set of all branches of \mathcal{X} can be enumerated as follows:

$$(3.12) \quad \text{branches}(\mathcal{X}) = \{X \subseteq \mathcal{X} : \text{isconvex}(\text{root}(X))\},$$

where $\text{root}(X) = \bigcup_{x \in X} x$,⁵ and the *isconvex* function tests whether a set is convex according to the definition in Section 2.3.1. The roots of all branches correspond to the nodes in the tree diagram for \mathcal{X} . A *pruned subtree* of \mathcal{X} can now be defined as the roots of a set of branches that partition \mathcal{X} . All possible pruned subtrees can be enumerated as follows:

⁵Observe that the root of the tree as a whole satisfies this definition, $\text{root}(\mathcal{X}) = \bigcup_{x \in \mathcal{X}} x = \phi_{\text{all}}(\mathcal{S})$.

$$(3.13) \quad \text{pruned}(\mathcal{X}) = \{\{\text{root}(X) : X \in \mathbb{X}\} : \mathbb{X} \subseteq \text{branches}(\mathcal{X}) \wedge \bigcup_{X \in \mathbb{X}} X = \mathcal{X} \wedge \bigcap_{X \in \mathbb{X}} X = \emptyset\}.$$

Every member of $\text{pruned}(\mathcal{X})$ is itself a tree.

The objective of the pruning stage of tree induction is to find the most efficient member of $\text{pruned}(\mathcal{X})$, where efficiency is a joint function of loss and parameter count. Since parameter count scales with the number of leaves in a tree, the natural efficiency measure for our purposes is a weighted sum of impurity and leaf count. In this case, the objective of pruning is to find

$$(3.14) \quad \mathcal{X}_\alpha = \underset{\mathcal{X}' \in \text{pruned}(\mathcal{X})}{\text{argmin}} \left[\sum_{x \in \mathcal{X}'} \frac{|\mathcal{D}_x|}{|\mathcal{D}|} \cdot \text{I}^A(\mathcal{D}_x) + \alpha |\mathcal{X}'| \right],$$

for some weighting parameter $\alpha > 0$, with ties between multiple optimal subtrees broken by selecting the smallest. The *minimal cost complexity pruning* (MCCP) algorithm [32] is able to solve this problem exactly. MCCP iteratively prunes one branch at a time, selected by the following greedy criterion:

$$(3.15) \quad \underset{X \in \text{branches}(\mathcal{X})}{\text{argmin}} \left[\frac{|\mathcal{D}_{\text{root}(X)}| \cdot \text{I}^A(\mathcal{D}_{\text{root}(X)}) - \sum_{x \in X} |\mathcal{D}_x| \cdot \text{I}^A(\mathcal{D}_x)}{|X| - 1} \right].$$

The iterative nature of MCCP means that it produces a sequence of pruned subtrees $\mathcal{X}, \mathcal{X}', \mathcal{X}'', \dots, \{\text{root}(\mathcal{X})\}$, each a subtree of the last, representing a progressive reduction of \mathcal{X} down to its root. The maximum length of this sequence is $m - 1$. Crucially, although the criterion in Equation 3.15 is greedy, the pruning sequence is guaranteed to contain the smallest optimally efficient subtree \mathcal{X}_α for every $\alpha > 0$ (the proof of this property is rather involved, so we refer the reader to Section 10.2 of [32]). Consequently, the number of options worth considering can be narrowed from the entire nested collection of pruned subtrees to the (at most) $m - 1$ members of the pruning sequence. Since the tree size regulariser α is a free parameter, any tree in the sequence could feasibly be chosen to define the policy model π' , in which case the final feature set ϕ would simply be the subset of ϕ_{all} that are used at least once in a rule in that tree. However, in our experiments, we explore the efficiency tradeoff between the model's fidelity on one end, and its interpretability (through simplicity) on the other, by selecting and evaluating several trees from across the pruning sequence.

One major advantage of the procedure described in this subsection is that it effectively reduces the joint problem of learning ϕ and π' to an iterative optimisation over impurity and leaf count. This exploits the fact that the growth and pruning of a tree is an exercise in feature selection as well as one of model building. Extraneous features that are not helpful for efficiently reducing impurity are ignored or pruned and can safely be discarded from the feature space. They thus have no detrimental effect on the interpretability of the final model. This in turn means that there is little risk in generating a very large initial feature space ϕ_{all} , with the primary cost to enlarging it being a (linear) increase in the time complexity of the tree growth stage. In practice, generating a large and unspecialised feature space may be less challenging than generating a smaller one containing only relevant features.

3.3 Traffic Simulator Experiments

We now demonstrate our approach in a simple traffic simulator environment, in which a population of vehicles are each controlled by a common black box policy to navigate around a track while avoiding collisions. Since the policy is homogeneous across the population, we can observe the behaviour of each vehicle independently to learn ϕ and π' .

3.3.1 Traffic Simulator Environment

The central structure of the traffic environment is a closed loop in 2D space, which self-intersects at one or more locations called *junctions*. Vehicles are modelled as rectangles and are constrained to move around the track longitudinally in a common direction (analogous to ‘slot car’ racing with a single slot). This environment has an arbitrarily large number of track shape variants, which we call *topologies*. All topologies considered in this work are constructed from line segments and circular arcs; five examples are shown on the left of Figure 3.3.

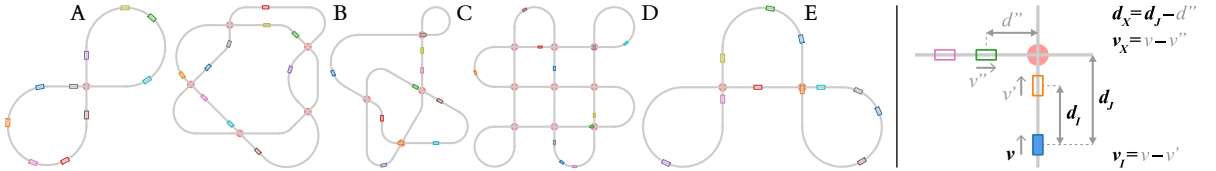


Figure 3.3: Left: A selection of track topologies for the traffic environment, with vehicles shown as coloured rectangles and junctions represented by red circles. Right: Definition of the six features used by π_F , as measured by the blue vehicle.

At the start of a simulation episode, the vehicle population \mathcal{I} is initialised by placing them at random points around the track (subject to a minimum-spacing constraint). By default, all vehicles start with a speed of v_{\max} , which is a fixed global speed limit for the environment. At time t , the state of the environment includes the position p_t^i of each vehicle $i \in \mathcal{I}$, which can be represented by a single real number: $p_t^i \in [0, L]$ where L is the overall length of the track. It also includes each vehicle’s speed $v_t^i \in [0, v_{\max}]$, as well as some representation of the track topology. For our purposes, we require a record of the total length of the track L , as well as the set of junctions \mathcal{J} . Each $j \in \mathcal{J}$ is associated with a location on the track, $p^j \in [0, L]$, as well as a *twin* $j' \in \mathcal{J}$, which is the other track location that intersects the first, $p^{j'} \in [0, L]$. For example, in topology A in Figure 3.3, \mathcal{J} has two elements specifying locations on the horizontal and vertical track sections that intersect to form the single junction.

The discrete action space \mathcal{A} consists of an ordered list of acceleration levels, whose values are symmetric about 0, and include 0 itself to allow a constant speed to be maintained. If at time t , vehicle i produces an acceleration action a_t^i , the environment responds by deterministically adjusting i ’s speed within the hard limits $[0, v_{\max}]$ for the following timestep, i.e. assigning $v_{t+1}^i \leftarrow \min\{\max\{v_t^i + a_t^i, 0\}, v_{\max}\}$. In all experiments discussed below, the action space is $\mathcal{A} = (-\frac{v_{\max}}{4}, -\frac{v_{\max}}{12}, 0, \frac{v_{\max}}{12}, \frac{v_{\max}}{4})$ with $v_{\max} = 0.1$, which is half the body length of a vehicle.

When using tree-based policy models to generate textual explanations later in this chapter, we represent these actions with symbolic labels ($--, -, 0, +, ++$).

The environment is capable of detecting collisions between vehicles and terminating the episode when one occurs. We utilise this functionality during the evaluation of policy models.

3.3.2 Target Policies

We consider two collision-avoiding control policies as the targets of our policy modelling approach. Both are hand-coded functions, rather than being learnt by some optimisation process (e.g. RL). Performing this initial evaluation against known target policies is useful, as it allows us to compare the resultant trees to a known ground truth. It is important to stress that from the perspective of the modelling procedure, the policies are black boxes, as only their inputs and outputs are observed.

- The **fully-learnable policy** (π_F) is a simple rule set which uses the values of six numerical features as measured from the perspective of each vehicle. These features include the vehicle’s absolute speed (v), the distance to the next junction (d_J), the distance to the nearest vehicle in front (d_I), and the speed relative to that vehicle (v_I). The final two features involve looking back along the other section of the track leading into the next junction, finding the next vehicle that is approaching it, and computing both the relative speed (v_X) and relative distance-to-junction (d_X) with respect to that vehicle. Figure 3.3 (right) depicts the definition of these features geometrically. This policy is itself implemented as a nested sequence of if-then rules over single features, equivalent to an axis-aligned tree with 39 leaves.⁶ It can thus be reconstructed exactly by our method in principle, provided all six features are included in the maximal feature space ϕ_{all} .
- The **partially-learnable policy** (π_P) retains some of the logic from π_F , but considers additional nearby vehicles and selects acceleration actions using a proportional feedback controller. These changes yield significantly smoother motion, but *cannot be precisely replicated* by a tree of finite depth, hence this policy can only be approximately modelled.

3.3.3 Recursive Generation of the Maximal Feature Space

The first step in our modelling approach is to synthesise a large set of state features for use in tree growth. We use a set of operators \mathbb{O} that enables all six features used by the fully-learnable policy π_F to be generated, since this ensures perfect replication of that policy is possible in principle. These operators are as follows:

- $\text{pos} : \mathcal{I} \rightarrow [0, L]$ or $\mathcal{J} \rightarrow [0, L]$. For a vehicle identifier $i \in \mathcal{I}$ or junction identifier $j \in \mathcal{J}$, return the position of that entity on the track.

⁶A tree diagram representing this policy is shown in Figure 3.9 (b).

- $\text{speed} : \mathcal{I} \rightarrow [0, v_{\max}]$. For a vehicle i , return its speed.
- $\text{fa} : \mathcal{I} \rightarrow \mathcal{I}$ or $\mathcal{J} \rightarrow \mathcal{I}$. For a vehicle i or junction j , return the next vehicle in front.
- $\text{ba} : \mathcal{I} \rightarrow \mathcal{I}$ or $\mathcal{J} \rightarrow \mathcal{I}$. For a vehicle i or junction j , return the next vehicle behind.
- $\text{fj} : \mathcal{I} \rightarrow \mathcal{J}$. For a vehicle i , return the next junction in front.
- $\text{twin} : \mathcal{J} \rightarrow \mathcal{J}$. For a junction j , return its twin j' .
- $\text{sep} : [0, L]^2 \rightarrow [0, L]$. For two track positions p and p' , compute how far p is in front of p' . The result is always non-negative since the track is a loop.
- $\text{sub} : [0, L] \rightarrow \mathbb{R}$ or $[0, v_{\max}] \rightarrow \mathbb{R}$. For two speeds v and v' or two separations sep and sep' , subtract the second from the first. The result can be negative.

The scalar values returned by three of these operators (speed , sep , sub) are taken as features; the remaining operators are used only to produce intermediate variables. The roles of these operators are best understood by considering how the six features used by π_F can be generated for a given vehicle $i \in \mathcal{I}$:

$$\begin{aligned}
 v &= \text{speed}(i); \\
 d_J &= \text{sep}(\text{pos}(\text{fj}(i)), \text{pos}(i)); \\
 d_I &= \text{sep}(\text{pos}(\text{fa}(i)), \text{pos}(i)); \\
 v_I &= \text{sub}(\text{speed}(i), \text{speed}(\text{fa}(i))); \\
 v_X &= \text{sub}(\text{speed}(i), \text{speed}(\text{ba}(\text{twin}(\text{fj}(i))))) ; \\
 d_X &= \text{sub}(\text{sep}(\text{pos}(\text{twin}(\text{fj}(i))), \text{pos}(\text{ba}(\text{twin}(\text{fj}(i))))) , \text{sep}(\text{pos}(\text{fj}(i)), \text{pos}(i))).
 \end{aligned}$$

All six features are generated by applying the operators to a maximum recursion depth of $r = 6$, but many additional features are created in the process. With the operators, domains and codomains as described, a total of 251,263 unique features are realisable with $r = 6$ (this number accounts for the fact that many feature combinations are redundant, e.g. $\text{ba}(\text{fa}(i)) = i$). To make the problem more manageable, we add a handful of additional constraints, including not applying the same operator twice in a row and always computing separations between vehicles and junctions (not vice versa), which reduce the number of candidate features to 308. This defines the maximal feature space ϕ_{all} . The vast majority of features remain irrelevant for reconstructing the target policies, so a challenging feature selection problem remains.

3.3.4 Dataset and Training

To produce the state-action dataset \mathcal{D} for a given target policy, we run that policy with a population of 11 vehicles on five different track topologies (specifically those shown in Figure 3.3). We run for episodes of 100 timesteps before resetting, and move to the next topology only when each of the five actions in \mathcal{A} has been used at least 5000 times across the vehicle population.

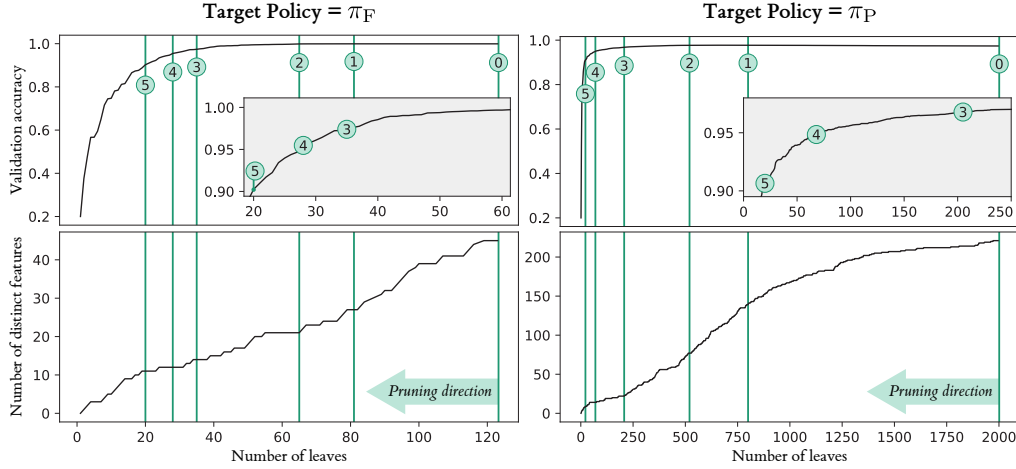


Figure 3.4: Curves of validation leaves set accuracy and number of features used across the pruning sequences for each target policy. The rightmost point in each curve corresponds to the unpruned tree \mathcal{X} . Numbered vertical lines indicate the trees chosen to carry forward to evaluation.

The recorded history of environmental states and agent actions is preprocessed by using the operators to generate the maximal feature vector ϕ_{all} for each vehicle i and storing this alongside the action produced by the policy for that vehicle. This means that each simulation timestep yields 11 feature vector-action pairs. We balance the action classes by randomly discarding data until each $a \in \mathcal{A}$ has the minimum count of 25000 observations (5000×5 topologies). Therefore, \mathcal{D} contains a total of 125000 observations (25000×5 actions).

During growth and pruning of the tree model, we use a pairwise loss function which penalises the absolute numerical distance between two actions: $\ell(a, a') = |a - a'|$. This creates an impurity measure which is effectively a distance-weighted version of the standard Gini impurity. Performing growth and pruning as described in Section 3.2.4 yields the sequence of pruned subtrees $\mathcal{X}, \mathcal{X}', \mathcal{X}'', \dots, \{\text{root}(\mathcal{X})\}$.

To determine which of these trees to take forward to evaluation, we measure their predictive accuracy on a validation dataset \mathcal{D}_{val} , of identical size and composition to \mathcal{D} . Accuracy is measured as the proportion of times the action of an observation in \mathcal{D}_{val} matches the modal action for the leaf that contains it, i.e. $\mathbb{E}_{(s,a) \in \mathcal{D}_{\text{val}}} [a = \text{argmax}_{a' \in \mathcal{A}} \pi'(a'|x) : \phi(s) \in x]$. We also consider the number of leaves and number of distinct features used in rules for each tree as heuristics for interpretability. Figure 3.4 shows curves for validation accuracy and feature count as a function of leaf count (which is unique for each tree in the pruning sequence) for both target policies. For each, we use an intuitive judgement to select five pruning levels in the sequence (in addition to the unpruned tree) that provide a reasonable spread across the fidelity-interpretability tradeoff. These are indicated in the figure by numbered vertical lines. Level 0 corresponds to the unpruned tree.

3.3.5 Baselines

To assess the effectiveness of the proposed approach, we compare it to several baselines. Firstly, we grow and prune a tree using only the six features required by π_F (denoted ϕ_F), thereby bypassing the feature generation/selection aspect of the problem. We also train another tree with a different set of hand-engineered features (denoted $\phi_{\text{naïve}}$), chosen to be indicative of what might be naïvely deemed useful without careful consideration or domain knowledge. These features are the radius, angle, normal velocity and heading of the two nearest vehicles (using the controlled vehicle’s egocentric coordinate system), as well as the controlled vehicle’s own speed v . Such measurements are blind to the locations of junctions and the topology of the track, so should be expected to yield lower-quality reconstruction. A second class of baselines is generated by following the full modelling procedure with ϕ_{all} , but using data from only one of the five track topologies at a time. These baselines assess the capacity of ϕ and π' to generalise between topologies, and whether training on multiple topologies is important for preventing overfitting. For each baseline, we use the single tree with the highest validation accuracy from the pruning sequence. Table 3.1 summarises the model variants considered in our experiments.

Table 3.1: Experimental model variants. Track topology letters correspond to those in Figure 3.3. * = per-topology training and multiple pruning levels only used with ϕ_{all} .

Property	Variants					
Target Policy	π_F			π_P		
Feature Set	ϕ_{all}		ϕ_F		$\phi_{\text{naïve}}$	
Training Topology*	All	A	B	C	D	E
Tree Pruning Level*	0	1	2	3	4	5

3.4 Quantitative Evaluation

Depending on how a policy model is intended to be used, its quality can be measured in various ways. A model with high predictive accuracy on a static, class-balanced dataset may exhibit rather different behaviour to the original agent when deployed as a policy in the underlying task environment, due to any distributional differences being amplified over time [211]. In addition, even if that behaviour were similar in most cases, environments tend to include certain critical states where selecting the optimal action is especially important if the policy is to be safe and performant. In the traffic environment, critical states arise in instances of high traffic density around junctions, where the risk of a crash is high. In this section, we evaluate tree-structured traffic policy models on a range of metrics to capture these various considerations, before moving on to explore how they may be used to interpret their black box targets in Section 3.5.

3.4.1 Evaluation by Predictive Accuracy

The first and most straightforward metric of policy model quality is its ability to predict the actions in an unseen test set $\mathcal{D}_{\text{test}}$. We use a test set with an identical size and composition to

the training set \mathcal{D} . The heat maps in Figure 3.5 show accuracy results for all model variants and baselines, both separated by test topology and aggregated across the whole test set. A great deal of information is carried by these visualisations, but the most salient points are:

- For both π_F and π_P , mean accuracy exceeds 90% for even the smallest pruning levels (i.e. fewest leaves) and 95% for the second-smallest.
- Accuracy for π_P is bounded at around 97.5%, reflecting the fact that this policy is not a tree so cannot be perfectly modelled by one. As expected, providing ϕ_F upfront (bypassing the feature selection problem) yields somewhat better accuracy, but it is promising to see that down to pruning level 2, accuracy differs by under 0.25% for both targets.
- Lacking information about junctions, the tree using $\phi_{\text{naïve}}$ is unable to obtain the same levels of accuracy, demonstrating the importance of choosing the correct set of features for policy modelling.
- The single-topology training results show that trees generalise well from the larger and more diverse topologies (especially C), but poorly from the small topology A. This suggests the latter produces insufficient diversity of vehicle arrangements to capture all important aspects of the target policies.

3.4.2 Evaluation by On-policy Divergence

Since agent policies operate in a fundamentally dynamic context, a complementary test of quality comes by deploying the target policy and tree-based policy to control all vehicles in two identically-initialised environments and measuring the similarity of the resultant behaviour over time. Specifically, we do this by measuring how rapidly the environment’s state diverges. The divergence at each timestep is defined as the absolute difference in cumulative distance travelled between the two instances of each vehicle, averaged across the population of 11 vehicles. Completing 100 episodes of 1000 timesteps for each pairing of model variant and test topology, we find that divergence is approximately linear with time, so obtain a single estimate by fitting a least squares linear regression model. The values in Figure 3.6 are the regressor’s divergence predictions for a full 1000-timestep episode. Key results are as follows:

- The maximum possible divergence value is $1000 \times v_{\text{max}} = 100$. Even the worst divergence results on the most challenging topologies do not come close to this upper bound, indicating that some consistency of driving behaviour is exhibited by all models.
- As with accuracy, enforcing the correct features ϕ_F or incorrect features $\phi_{\text{naïve}}$ tends to have a large effect on divergence. With π_F as the target, models using the full feature set ϕ_{all} are competitive with using ϕ_F down to pruning level 2.
- With π_P as the target, the increase in divergence with pruning level is far more gradual (even nonexistent for some topologies). This result has interesting implications: while this

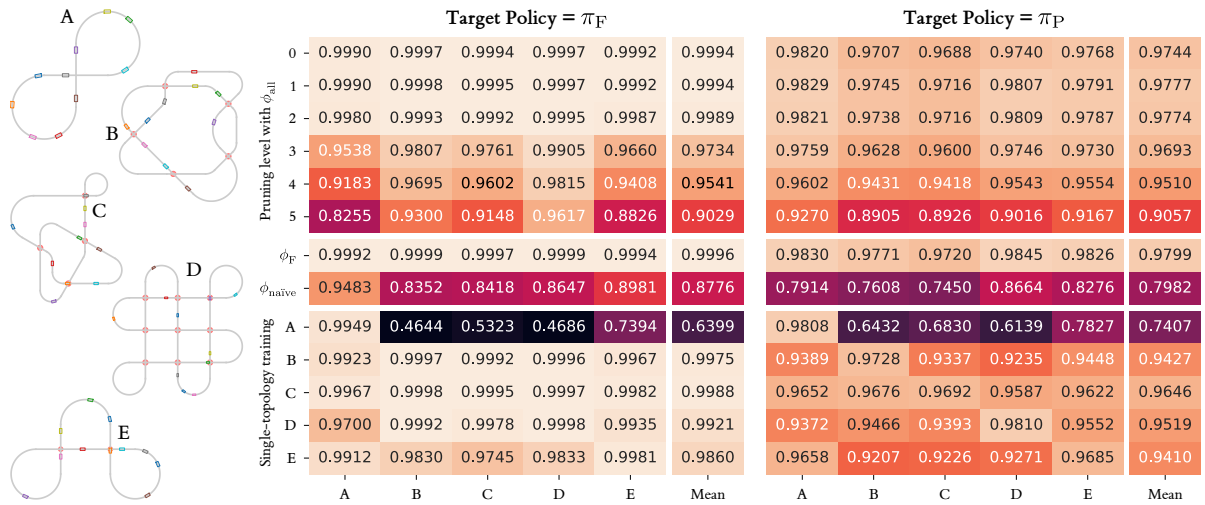


Figure 3.5: Left: Track topologies. Right: Model accuracy on an unseen test set $\mathcal{D}_{\text{test}}$.

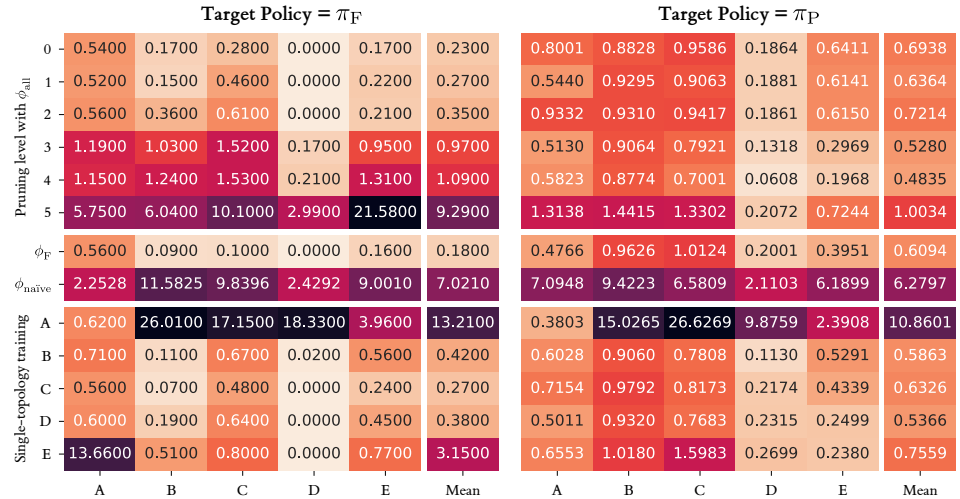


Figure 3.6: Expected divergence from target policy after 1000 timesteps.

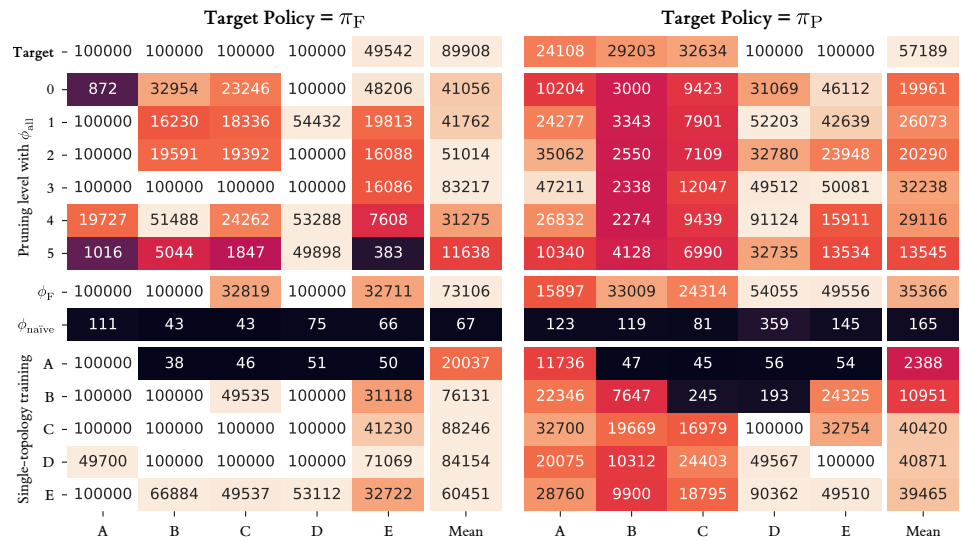


Figure 3.7: Mean time between failures across 100 episodes of up to 1000 timesteps.

policy is impossible for a tree to exactly replicate by design, it may be somewhat easier to imperfectly model it in a way that preserves similar on-policy behaviour.

- The clear column-wise banding of both matrices in this figure indicates that differences in the intrinsic difficulty of each topology are as influential as the differences between models. The longest (and thus least densely-populated) topology D is almost always the easiest to avoid diverging on, with the best models exhibiting no measurable divergence at all after 1000 timesteps. Uniformities in the inter-junction distances of this topology may also play a role.
- Intrinsic topology differences are especially clear in the submatrix for the single-topology baselines. Here, the on-diagonal elements are no larger than the off-diagonal ones, indicating that training on a given topology does not yield any special improvement in divergence on that topology.

3.4.3 Evaluation by Mean Time Between Failures

The final quality metric considered is the ability of the tree-based policy models to avoid certain failure conditions when deployed in the environment. We consider two modes of failure here: a collision between multiple vehicles, and a *stall*, which we define as three or more vehicles being stationary for 15 consecutive timesteps,⁷ thereby inhibiting traffic flow. We again evaluate using 100 episodes of 1000 timesteps with 11 vehicles. The values in Figure 3.7 are the mean times between failures (MTBF) by either of the two modes, which are cumulative across episodes (i.e. three complete episodes plus a failure after 100 timesteps yield an MTBF of 3100). The maximal value of 100000 is used when zero failures occur; this is a conservative figure since the policy may have continued for even longer without failing. The target policies π_F and π_P are included in this figure for comparison. Key results are as follows:

- While MTBF results broadly correlate with accuracy, this metric shows a more marked aggregate distinction between π_F and π_P , and between different test topologies.
- Nonetheless, there is minimal degradation with pruning down to level 4, with a constant average of just 1-2 failures for π_F , and 3-4 for π_P .
- In fact, it appears that intermediate pruning levels fail less often than the largest trees. While the reason for this is not immediately clear, it may be that having fewer leaves yields less frequent changes in acceleration and smoother motion.
- In rare cases (e.g. π_P pruning level 3, topology A), tree models may even fail less often than their target policies.
- Providing ϕ_F confers no significant benefit over ϕ_{all} , while $\phi_{\text{naïve}}$ and single-topology training on A are utterly unable to perform and generalise well respectively.

⁷Except in topology A, which is the smallest and most densely-packed. Here, periods of stop-start motion are unavoidable, and we register a stall only when *all* vehicles are stationary for 15 timesteps.

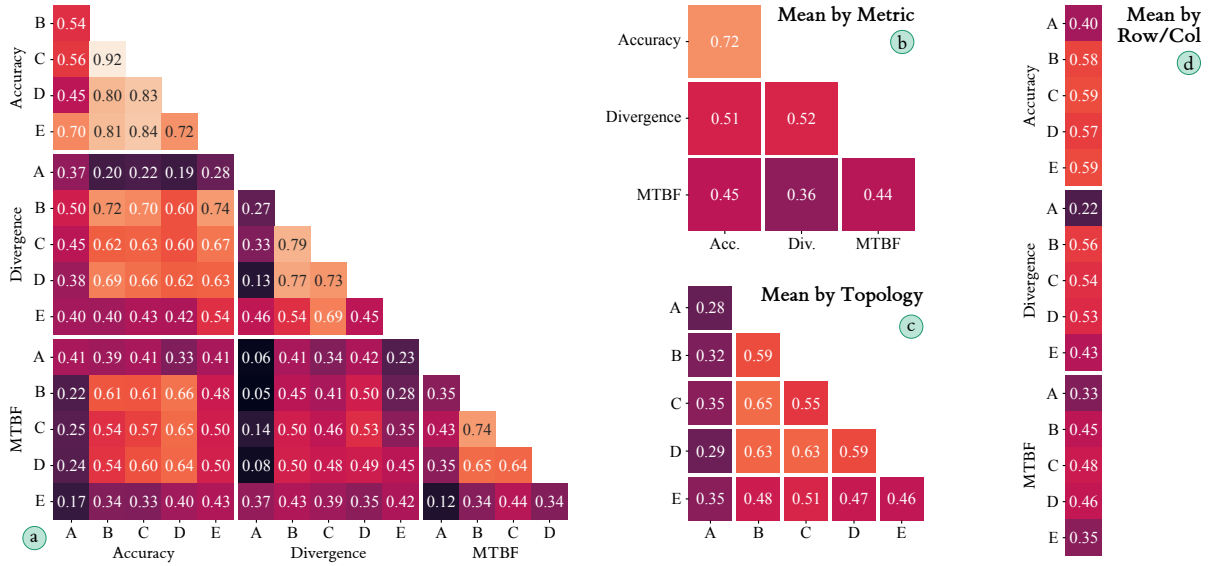


Figure 3.8: Correlations between model rankings on all topologies and quality metrics.

3.4.4 Correlations between Quality Metrics

We now assess the relationships between the quality of all 26 models (2 target policies \times (6 pruning levels + 7 baselines)) across the various metrics and topologies. Since we already have evidence that these relationships are nonlinear (e.g. small accuracy changes leading to large differences in MTBF), we quantify them using Kendall's τ rank correlation coefficient [139] instead of Pearson correlation. This tells us how much a model's rank (out of 26) on one metric-topology pair (e.g. accuracy on topology C) correlates with its rank on another metric-topology pair (e.g. divergence on topology A). Pairwise correlations between all metrics, on all topologies, are shown as a lower triangular matrix in Figure 3.8 (a). All values in this matrix are positive, with the mean value being 0.47, indicating that knowledge of a model's accuracy, divergence or MTBF on one topology provides at least some signal about its efficacy in other contexts.

To highlight the key trends, we group and average the correlation values by metric (b) and topology (c). In the first of these aggregated matrices, the accuracy-accuracy correlation is highest at 0.72, indicating that a model's accuracy rank on one topology is fairly strongly predictive of its accuracy on another (this value increases to 0.82 if topology A is excluded). Moving down the rows, the on-policy metrics of divergence and MTBF are more independent both of each other, and of their respective values on other topologies. (c) highlights the single-junction topology A as an outlier in terms of cross-topology generalisation, with the three most complex topologies (B, C and D) being somewhat more mutually predictive than when E is also included. This reinforces that robustly assessing policy models requires diverse evaluation data.

Finally, (d) shows row/column-wise means of the correlation matrix, of each metric-topology pair with respect to all others. This shows that high accuracies on topologies C and E are the best overall indicators of a successful model. The least generally predictive metric is on-policy

divergence on topology A; merely assessing a model by this metric would tell us very little about its wider quality. Overall, this correlation exercise serves to highlight how policy models can (and should) be evaluated from a multitude of perspectives, which interact and correlate in subtle ways. We can identify no single metric that paints a complete picture.

3.5 Model Interpretation and Explanation

The preceding analysis evaluates our proposed framework for tree-based policy modelling according to various quantitative metrics, but tells us nothing about the utility of the models for understanding their target policies. This section discusses various ways in which a tree can be used to summarise, explain and diagnose target policy behaviour. Favouring depth of exploration over breadth, we focus exclusively on one model: the most heavily-pruned tree (level 5) for the fully-learnable policy π_F , since it is compact enough to visualise in full for didactic purposes. This model achieves a mean predictive accuracy of just over 90% on a class-balanced test set (see Figure 3.5). While this level of accuracy is sufficient to provide certain kinds of insight, it clearly does not indicate a perfect model. Since we are in the artificial situation of knowing the true π_F exactly, we can discuss the ways in which the model’s rules and explanations are faithful, and ways in which they are potentially misleading.

3.5.1 Tree Diagram Comparison

As discussed in Chapter 2, a key factor in the interpretability of tree models is the ability to visualise their rule structures as diagrams. Figure 3.9 shows the tree diagram of the level 5 policy model for π_F (a), alongside the ground truth tree structure of π_F itself (b). In both cases, each leaf of the tree is associated with an action (the modal action, for the learnt tree) and the hierarchy of rules determines which leaf is reached based on features of the environment state. If a rule evaluates to true, the right branch below is taken. Otherwise, the left branch is taken. The diagrams provide a complete description of the target and model policies, which can be directly compared.

The learnt model has 20 leaves and utilises 11 of the 308 features from ϕ_{all} in its rules. Among these are all six features used by the target policy. We see this as being a very positive result, indicating that the tree growth and pruning process is capable of singling out the features required to faithfully reconstruct a black box policy from a much larger set of alternatives. The remaining five features include the distance between the vehicle in front and the junction in front ($d_J - d_I$, rule shown in yellow box), which is a composite of two of the correct features. Also included is a relatively complicated construction that ultimately just computes $-d_I$ (used in two rules, shown in blue boxes). This feature is a redundant addition because d_I is already present, but since its rules could easily be inverted to eliminate it, its inclusion does not create problems for interpretation. However, the tree also utilises three more spurious features that are

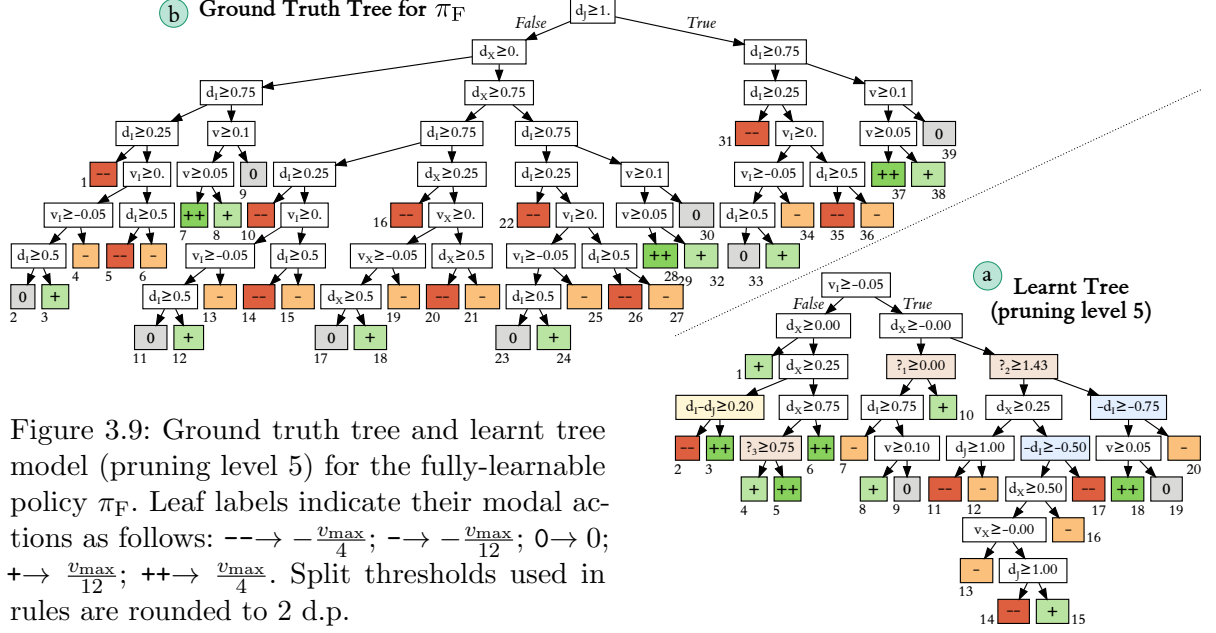


Figure 3.9: Ground truth tree and learnt tree model (pruning level 5) for the fully-learnable policy π_F . Leaf labels indicate their modal actions as follows: $-- \rightarrow -\frac{v_{\max}}{4}$; $- \rightarrow -\frac{v_{\max}}{12}$; $0 \rightarrow 0$; $+ \rightarrow \frac{v_{\max}}{12}$; $++ \rightarrow \frac{v_{\max}}{4}$. Split thresholds used in rules are rounded to 2 d.p.

significantly different from those used by the target policy (denoted $?_1$, $?_2$ and $?_3$, rules shown in orange boxes). The derivations of these features in terms of the canonical operators are:

$$\begin{aligned} ?_1 &= \text{sub}(\text{speed}(\text{fa}(i)), \text{speed}(\text{ba}(\text{fj}(i))))); \\ ?_2 &= \text{sub}(\text{sep}(\text{pos}(\text{fj}(i)), \text{pos}(i)), \text{sep}(\text{pos}(\text{ba}(\text{fj}(\text{fa}(i)))), \text{pos}(\text{fa}(i)))); \\ ?_3 &= \text{sub}(\text{sep}(\text{pos}(\text{ba}(\text{fj}(i))), \text{pos}(i)), \text{sep}(\text{pos}(\text{ba}(\text{fj}(\text{fa}(i)))), \text{pos}(\text{fa}(i)))). \end{aligned}$$

For example, $?_1$ can be interpreted as the relative speed between the vehicle in front, $\text{fa}(i)$, and the first vehicle behind the upcoming junction, $\text{ba}(\text{fj}(i))$ (note that these might be the same vehicle, in which case $?_1 = 0$). One could construct a plausible story for why this might be useful for a control policy, but to the extent that $?_1$ influences the tree model's output, this reflects correlations in the state-action dataset \mathcal{D} rather than truly causal mechanisms within the target policy. Rules using $?_1$, $?_2$ and $?_3$ may have value for understanding trends in the agent's behaviour but must be interpreted with significant caution.

For the six correct features, it is informative to look at the thresholds involved in rules in the learnt tree. For example, for features measuring relative distances to other vehicles (d_I and d_X), the thresholds 0.25, 0.5 and 0.75 appear often, and for the distance-to-junction feature (d_J), a threshold of 1 is used on both occasions. Inspecting the ground truth tree reveals that these are precisely the thresholds used when hand-coding the target policy π_F . The model has thus not only selected the correct features, but made use of them in largely the correct ways, thereby accurately reconstructing some of the internal logic of the target policy. That said, the order in which rules are combined differs (e.g. the first rule in the ground truth tree tests whether $d_J \geq 1$, but this rule is used further down in the learnt tree). This indicates that the tree learning process has somewhat imprecisely encoded the circumstances in which each rule is applied. Such imprecision contributes to the 10% predictive error rate of this policy model.

3.5.2 Factual Local Explanation

In addition to providing an approximate global overview of how the black box target policy operates, a tree model can be used to give a symbolic local explanation of an individual action in terms of the rules applied to produce it, provided the model’s prediction matches the action actually taken (no meaningful factual explanation exists if the two actions are inconsistent). In this case, we take the modal action of each leaf as the prediction, which equates to interpreting the tree as a deterministic policy. The symbolic explanation is constructed via a process of *ancestor rule conjunction*, which is first introduced in Section 2.3.4. For example, suppose at a particular timestep the feature vector for vehicle i ends up at the leaf numbered 1 in Figure 3.9 a. This leaf has two ancestors, whose rules are “ $v_I \geq -0.05$ ” and “ $d_X \geq 0$ ” (both evaluate to false for this leaf), and the modal action is a soft acceleration (+). If this matches the policy’s true action, the following factual explanation can be offered:

“ i ’s action is + because $v_I < -0.05$ and $d_X < 0$ ”,

i.e. the policy chooses to accelerate because the vehicle in front is going at least 0.05 faster than it, and the first vehicle on the opposite branch into the next junction is currently further than it from the junction. In informal terms, i ‘seizes the opportunity’ to accelerate into a gap before the opposite vehicle crosses the junction.

Recall from Section 2.3.4 that if the same feature is tested multiple times along the path of ancestors, it is only necessary to state the most restrictive condition. For example, an observation that ends up at leaf 6 can be explained by

“ i ’s action is ++ because $v_I < -0.05$ and $d_X \geq 0.75$ ”,

which omits two other ancestor rules that are subsumed by “ $d_X \geq 0.75$ ”.

Since in this work, we are in the artificial position of knowing the internal structure of the target policy π_F , it is also possible to identify when such explanations are misleading. The factual explanation of an observation at leaf 10 is:

“ i ’s action is + because $v_I \geq -0.05$ and $d_X < 0$ and $?_1 \geq 0$ ”.

This includes the spurious feature $?_1$, which is used nowhere in the decision logic of π_F but has nonetheless been identified as informative during the tree growth process because its value correlates with the policy’s action in the training set. Provided the prediction of + does match the target policy for this observation, then we can conclude that the model has given the *right answer*, but for (at least partially) the *wrong reason*. This example serves to show how the non-equality of correlation and causation is especially relevant when it comes to generating explanations using approximate models [119]. One possible way to make a factual explanation less causally loaded is by replacing the word “because” with “which is reliably predicted by”.

3.5.3 Counterfactual Local Explanation

Presented in isolation, factual explanations such as those given above have a somewhat shallow quality, since they lack any reference to feasible alternatives. When a person asks for an explanation of an event, they are often implicitly seeking a reason why a different event, known as a *foil*, did not occur instead [160]. In the context of agent decision making, a *class-contrastive counterfactual explanation* provides reasons why an agent’s true action a^* differs from a specified foil action a' . The tree models used in our approach provide an elegant mechanism for class-contrastive counterfactual explanation.

Consider a state s^* for which the tree correctly predicts the target policy’s action a^* , i.e. $\phi(s^*)$ lies within leaf $x^* : \operatorname{argmax}_{a \in \mathcal{A}} \pi'(a|x^*) = a^*$. We may wish for a counterfactual explanation as to why another action $a' \neq a^*$ is not taken instead. The first step is to enumerate all conditions that are *sufficient* for the foil to be satisfied, which in a tree are the leaves for which a' is the modal action:

$$(3.16) \quad X_{(\text{action}=a')} = \{x \in \mathcal{X} : \operatorname{argmax}_{a \in \mathcal{A}} \pi'(a|x) = a'\}.$$

Any feature vector $\mathbf{f}' \in \bigcup_{x \in X_{(\text{action}=a')}} x$ is a sufficient counterfactual for the foil action a' . However, the literature on counterfactual explanations for machine learning places a heavy emphasis on the principle of *minimality*: we should provide not just any sufficient counterfactual, but one that requires the smallest possible change to the original features $\phi(s^*)$ [102, 261].

Minimality is best understood with reference to the space partitioning representation of trees. Recall that the leaves of a tree \mathcal{X} are axis-aligned hyperrectangles in the feature space. For each $x \in \mathcal{X}$, let $\text{lo}(x, d)$ and $\text{hi}(x, d)$ be the lower and upper boundaries of the hyperrectangle along feature d , which are defined by the splits made at its ancestors.⁸ As a shorthand, let us denote $\mathbf{f}^* = \phi(s^*)$ to be the feature vector for the original state. The location in x which is closest to \mathbf{f}^* , denoted by $\mathbf{f}_d^{(* \rightarrow x)}$, can be defined on a feature-wise basis:

$$(3.17) \quad \mathbf{f}_d^{(* \rightarrow x)} = \begin{cases} \text{hi}(x, d) & \text{if } \mathbf{f}_d^* > \text{hi}(x, d), \\ \text{lo}(x, d) & \text{if } \mathbf{f}_d^* < \text{lo}(x, d), \\ \mathbf{f}_d^* & \text{otherwise,} \end{cases} \quad \forall d \in \{1, \dots, D\}.$$

Because leaves are hyperrectangles, this closest-point definition is unambiguously true for any p -norm. For each $x \in X_{(\text{action}=a')}$, let $\delta^{(* \rightarrow x)} = (\mathbf{f}^{(* \rightarrow x)} - \mathbf{f}^*)$ be the vector from \mathbf{f}^* to $\mathbf{f}^{(* \rightarrow x)}$. Practically speaking, $\delta^{(* \rightarrow x)}$ describes the smallest changes to the feature values in \mathbf{f}^* that would move it from x^* to x . Together, the δ vectors provide a set of locally-minimal alternatives for how the original state could be modified such that the tree would predict the foil action a' .

We could stop here and return the full set of closest points as a counterfactual explanation. In some contexts, this may be desirable as a diverse counterfactual set helps to build a more

⁸One or both of these boundaries will effectively be infinite if none of the ancestors are split along d , but this creates no problems for counterfactual generation.

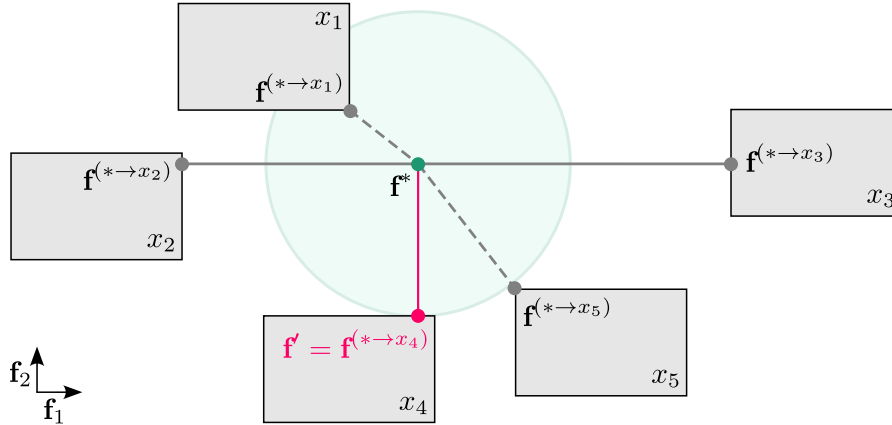


Figure 3.10: To find the minimal counterfactual for state s^* and foil action a' , we start by identifying the set of leaves satisfying the foil condition $X_{(\text{action}=a')}$ ($= \{x_1, \dots, x_5\}$ here), then find the closest point to $\mathbf{f}^* = \phi(s^*)$ on the boundary of each. We filter first by the 0-norm of the corresponding δ vectors; this removes $\mathbf{f}^{(* \rightarrow x_1)}$ and $\mathbf{f}^{(* \rightarrow x_5)}$ from contention. We then sort by 2-norm (faint green circle), which identifies $\mathbf{f}^{(* \rightarrow x_4)}$ as the minimal counterfactual.

well-rounded understanding of the policy [183]. In others, however, it may risk overloading a human with too much information. For this reason, we now propose a simple two-stage method for selecting the single minimal element of the set $\{\mathbf{f}^{(* \rightarrow x)} : x \in X_{(\text{action}=a')}\}$ by considering the corresponding δ vectors.

The closest element in the set to \mathbf{f}^* depends on which norm is applied to the δ s, and different norms have different advantages. The 2-norm (Euclidean distance) corresponds to the intuitive notion of distance in vector spaces, especially those with a physical interpretation, but does not incentivise sparsity, which would allow us to give a more compact explanation (fewer feature changes means fewer clauses in the textual output). Conversely, the 0-norm only measures sparsity, and cannot differentiate between options with the same number of nonzero elements. The 1-norm (Manhattan distance) offers a popular compromise [183, 261], but we propose a different approach, which is summarised in Figure 3.10. We filter first by 0-norm for sparsity,

$$(3.18) \quad X_{(\text{action}=a' \wedge \text{sparse})} = \{x : \|\delta^{(* \rightarrow x)}\|_0 = \min\{\|\delta^{(* \rightarrow x')}\|_0 : x' \in X_{(\text{action}=a')}\}\},$$

then by 2-norm to find the minimal counterfactual (ties broken arbitrarily):

$$(3.19) \quad \mathbf{f}' = \mathbf{f}^{(* \rightarrow x)} : \|\delta^{(* \rightarrow x)}\|_2 = \min\{\|\delta^{(* \rightarrow x')}\|_2 : x' \in X_{(\text{action}=a' \wedge \text{sparse})}\}.$$

This two-stage approach enforces a strict priority of the 0-norm over the 2-norm; a counterfactual is only considered if its δ vector is at least as sparse as any of the others. We suggest that this is desirable behaviour, because any features whose values do not need to change for \mathbf{f}^* to be reassigned the foil action can be excluded from the explanation. Prioritising the 0-norm puts the strongest possible emphasis on compact explanations, while still punishing δ vectors with very large (Euclidean) magnitudes. It also makes it simple, if required, to specify a hard

threshold on sparsity, allowing us to answer questions of the form “can the foil condition be realised by changing $\leq n$ features?”.

Returning to the learnt tree in Figure 3.9 (a), consider a feature vector for vehicle i that ends up at leaf 10, for which the prediction is a soft acceleration (+). We may wish to know why a hard deceleration (--) is not performed instead. Unlike in factual explanation, the precise location of the vector within the leaf affects the explanation because it changes the distances to other leaves. If for this vector, $?_2 < 1.43$ and $d_J < 1$, the following counterfactual is minimal:

“ i ’s action would be -- if d_X increased to ≥ 0 (leaf 11)”,

i.e. all else being equal, the policy would have output a hard deceleration if the first vehicle on the other branch into the next junction were not further than it from the junction. In this case, this is the only counterfactual with one non-zero feature change. As another example, for a feature vector at leaf 16 (also subject to $d_X < 0.75$, $?_3 < 0.75$, $v \geq 0.05$) and a foil action of ++, there are three counterfactuals requiring two features to be changed. If $d_X < ?_3$ and $(v_I + 0.05)^2 + (d_X - 0.75)^2 < (v - 0.05)^2 + (?_2 - 1.43)^2$, the one in bold is minimal by 2-norm:

“ i ’s action would be ++ if
 v_I decreased to < -0.05 and $?_3$ increased to ≥ 0.75 (leaf 5)
or v_I **decreased to < -0.05 and d_X increased to ≥ 0.75 (leaf 6)**
or v decreased to < 0.05 and $?_2$ increased to ≥ 1.43 (leaf 18)”,

i.e. i would switch from a soft deceleration to a hard acceleration if the vehicle in front were going faster than it, and the distance to the next junction were larger.

As mentioned above, factual explanations cannot be generated when the tree’s prediction is inconsistent with the target policy. With counterfactuals, a trivial modification – setting the foil to be the target’s true action – allows us to shift the focus to the shortcomings of the model itself. For example, suppose a feature vector for vehicle i ends up at leaf 9, for which the prediction is no acceleration (0), but we know from observation that the policy actually performs a soft deceleration (-). The minimal counterfactual is to move to leaf 7:

“Model would match target if d_I decreased to < 0.75 ”,

or equivalently, if the threshold in the rule separating leaf 7 from leaves 8 and 9 were increased by the same amount. This result does not automatically tell us where the flaw in the model lies (perhaps the decision threshold is misplaced and should indeed be moved, or perhaps the model requires an additional rule), but it does point to parts of the feature space where we might learn from more experimentation with the target policy. This perspective treats the policy model not as an end in itself, but as a tool to be used during a wider effort to build intuition, understanding and trust in the black box policy.

3.5.4 Temporal Explanatory Stories

Much of the complexity of agent-based problems stems from the fact that they are dynamic, yet most prior efforts to use tree models for policy interpretation operate on a single-timestep basis [51, 186, 251]. This subsection presents a method for explaining a target policy’s behaviour over many timesteps, derived from the counterfactual analysis described above.

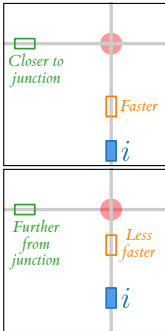
An explanation of a time period in a dynamical system is, in effect, a *story*, which should identify the salient events and their proximal causes. Where the system consists of an agent i interacting with its environment, a sensible choice for these salient events is the sequence of times when i changes its action. In the language of tree models, this means moving from one leaf x^t at time t to another leaf x^{t+1} at time $t+1$, such that $\operatorname{argmax}_{a \in \mathcal{A}} \pi'(a|x^t) \neq \operatorname{argmax}_{a \in \mathcal{A}} \pi'(a|x^{t+1})$. As long as the tree is consistent with the target policy in both timesteps, this transition can be explained by adapting the concept of a minimal counterfactual.

Given the feature vector for the state at time t , denoted by \mathbf{f}^t , Equation 3.17 can be used to find $\mathbf{f}' = \mathbf{f}^{(t \rightarrow x^{t+1})}$, the closest point inside x^{t+1} . This indicates the minimal feature value changes required to move to the new leaf. We suggest that it is also beneficial to explicitly state the features that must *not* change through the transition from x^t to x^{t+1} , since this provides some background context. These features are those tested along the decision path to x^{t+1} which have values of zero in the difference vector $\delta^{(t \rightarrow x^{t+1})}$. Consider the following example:

“ i ’s action changed from + (leaf 8) to 0 (leaf 9)
 because v increased to ≥ 0.1 ,
 while $v_I \geq -0.05$ and $d_X < 0$ and $?_1 < 0$ and $d_I \geq 0.75$ ”,

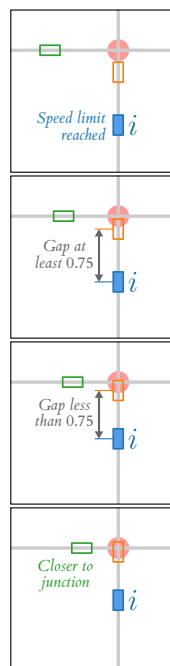
i.e. the policy stops accelerating because the vehicle has reached a speed of 0.1 (which is the speed limit v_{\max}), while the conditions on the third line remain constant.

To explain an extended sequence of behaviour, the preceding analysis can be performed iteratively, omitting any timesteps when the target policy’s action remains constant. We propose to start the explanatory story with a single factual explanation, then add a counterfactual whenever the action changes. In cases where the model’s prediction does not match the target, we deploy the aforementioned method of self-critiquing by using the true action as a foil. These ideas are all combined in the example below, which seeks to explain why a vehicle i moves from hard acceleration to hard deceleration over a period of 50 timesteps:



“ $t = 0$: i ’s initial action is ++ (leaf 6)
 because $v_I < -0.05$ and $d_X \geq 0.75$.”

“ $t = 8$: i ’s action changed from ++ to + (leaf 8)
 because v_I increased to ≥ -0.05 and d_X decreased to < 0 ,
 while $?_1 < 0$ and $d_I \geq 0.75$ and $v < 0.1$.”



“ $t = 17$: i ’s action changed from + to 0 (leaf 9)
 because v increased to ≥ 0.1 ,
 while $v_I \geq -0.05$ and $d_X < 0$ and $?_1 < 0$ and $d_I \geq 0.75$.”

“ $t = 32$: i ’s action changed from 0 to - but model still predicts 0 (leaf 9).
 Model would match target if d_I decreased to < 0.75 (leaf 7).”

“ $t = 39$: state moves to leaf 7 so model matches target again.”

“ $t = 50$: i ’s action changed from - to -- (leaf 17)
 because d_X increased to ≥ 0.25 ,
 while $d_I \leq 0.5$ ⁹ and $?_2 < 1.43$.”

This explanatory story tells us that from a starting point of hard acceleration, vehicle i initially lowers its acceleration ($t = 8$) because the speed relative to the vehicle in front has exceeded -0.05 (i.e. the gap between the vehicles is no longer growing significantly), and the closest vehicle on the opposite branch is now further than it from the junction. i then moves to a constant speed ($t = 17$) because the speed limit of 0.1 is reached. After this, the initial shift to a soft deceleration ($t = 32$) is not captured by the model, but agreement is recovered 7 timesteps later, and an explanation for the mismatch is provided. The final change to hard deceleration ($t = 50$) is explained by the vehicle on the opposite branch moving ahead of i by at least 0.25 units, requiring i to slow down to let the other vehicle cross the junction first.

Much of this story is aligned with the general functionality programmed into the target policy, and gives useful insight into its correlative properties. However, it is clear given our knowledge of the true tree structure of π_F (shown in Figure 3.9 (b)) that some of the above explanatory statements are more causally accurate than others. Although most of the decision thresholds used in the story (e.g. $v_I = -0.05$, $d_X = 0.75$, $v = 0.1$) are present in the true tree structure, they are combined in different ways that may yield differences in the counterfactual statements produced. More subtly, the preceding counterfactuals state the minimal change required to move from x^t to x^{t+1} (e.g. from leaf 6 to leaf 8), but *not* the minimal change to change the tree model’s action prediction (e.g. from ++ to +). They may not be equivalent, specifically in the case that x^{t+1} is adjacent (in the feature space) to leaves with the same predicted action. As a result, this method as stated is liable to produce overly restrictive, non-minimal counterfactuals for action change. This issue is revisited in Section 3.9.2, where we develop a refined method for temporal explanation that generates truly minimal counterfactuals.

⁹Here the redundant feature $-d_I$ has been removed by reinterpreting $-d_I \geq 0.5$ as $d_I \leq 0.5$.

3.6 Pit Stop: Reviewing the Policy Modelling Approach

3.6.1 Related Work

This chapter began by noting that interpretable policy modelling “*naturally springs to mind*” as a strategy for understanding black box agents, and indeed, the use of trees for this purpose is widespread. An early example is the *Cognitive Shadow* project [151], in which CART policy models are learnt from observational data in a simulated air defence command centre. The target policy is that of a human decision maker, and the main purpose of the model is to generate interpretable alerts if that person deviates from their historic behaviour. Subsequent papers by the same authors find that trees achieve superior fidelity to other interpretable models such as logistic regression and nearest neighbours clustering. More similar to our own work, [47] use CART to model a hand-coded “*exemplary*” policy in a driving simulator with various track topologies. Like us, they use on-policy failure rate as a complementary metric to fidelity, and observe poor cross-topology generalisation if the training data are not diverse. They propose to record failure cases in a supplementary dataset for selectively refining the tree structure.

Such iterative learning is also the focus of VIPER [20], which uses the DAGGER [211] strategy of retraining the policy model on data from its own on-policy distribution to gradually narrow the mismatch to the target policy. On each iteration, CART is used to grow a new tree from scratch on the enlarged dataset.¹⁰ Numerous works have built on the VIPER algorithm. For example, [212] complements the data-driven tree learning phase with a sequence of manual domain-specific rule fixes (e.g. to reduce undesirable oscillations), yielding an interpretable and high-performing tree policy, and [256] extends VIPER with a “*gating function*” that nonlinearly partitions the state space into regions where local tree policy models are learnt. This increases the potential expressivity of the aggregate model.

The latter work is one of many to depart from a basic axis-aligned tree structure with the aim of increasing fidelity or performance. Other examples are [58], which finds that allowing non-axis-aligned (oblique) partitions somewhat improves model quality on both metrics at the cost of significantly more parameters, and [71], which further generalises to nonlinear partitions and employs bilevel evolutionary optimisation. Yet others replace definite true/false rule outcomes with probabilistic ones, and refer to the resultant trees as *fuzzy* [251] or *soft* [51]. [192] use soft tree policy models in the medical domain, where they argue that trees are especially natural because of their pervasiveness in clinical education, and propose a novel recurrence mechanism to handle partial observability. While soft trees are more expressive in principle, and can be more easily learnt from streaming observations due to their differentiable parameters, their probabilistic semantics make interpretation far more difficult. [73] consider post-processing soft trees through discretisation, but find that this causes a large drop in fidelity.

¹⁰We explored using DAGGER in our own experiments but found no further performance improvement. This may be because our training dataset was sufficiently large and diverse to cover all relevant states.

[57] perform a detailed comparison of soft and hard (CART) trees for policy modelling in three benchmark control domains. Like us, they find that the relationship between model fidelity and performance is highly nonlinear. Furthermore, while soft trees are superior at small maximum depths, hard trees continue to improve with size while soft trees plateau, even as their parameter count grows exponentially (this observation is also made in [51]). It is therefore not clear that the more complex and expensive soft tree learning approach performs better than CART in general. A similar finding in favour of simple tree algorithms is made in [96], where a greedy CART-like approach is compared to more expensive mixed-integer programming and gradient descent methods for policy modelling in robotics domains.¹¹ Somewhat surprisingly, the greedy trees are often superior, which is attributed to them being able to model the full policy in a single tree rather than one per discrete action, their ability to be grown to a maximum leaf count rather than a fixed uniform depth, and their reduced hyperparameter sensitivity. These comparative studies bolster our own decision to base our algorithms on CART.

We also make the following general observations about tree-based policy modelling:

- Most work takes it as given that a fixed (and interpretable) feature set is available to use in tree growth, which is not always realistic. An exception is [73], which uses a secondary soft tree to learn a linear transformation of the state space prior to the main policy model. Our operator-based feature generation approach is an alternative to this method.
- Evaluations tend to focus primarily on fidelity/performance metrics, to the extent that demonstrations of the interpretability of the learnt trees are often missing entirely [58] or rather cursory and unconvincing due to their large size and/or nonlinear partitions [73]. In the preceding sections, as throughout the rest of this thesis, we have attempted to do equal justice to the quantitative and qualitative evaluation of interpretable models.
- It is common to optimise and evaluate trees for both fidelity and on-policy performance. Since these metrics are not equivalent, the ultimate purpose of the trees is often unclear: are they (1) models for understanding black box policies, or (2) standalone *white box* policies for deploying in the environment? Our own evaluation is somewhat guilty of this conflation, but in the context of this thesis, we can be clear that (1) is our aim.

The clear non-equivalence, and common conflation, of model fidelity and on-policy performance inspired significant further thought on the topic after this work was completed. This resulted in a discussion paper [26], in which we conclude that interpretable policy models should be framed *either* as tools for understanding their black box targets, *or* as usable policies in their own right, but not both simultaneously. This echoes recommendations made in [275] for the neural network rule extraction field.

¹¹In all cases, each leaf is associated with a linear predictive model rather than a summary distribution.

3.6.2 What is Missing in a Policy Model?

A more basic critique of the policy models proposed so far is that they are fundamentally incomplete. The approach is the natural analogue of learning an interpretable surrogate of a supervised learning model [180], but this very analogy risks masking the additional complexities present in the agent context. Contrary to a supervised learning system, which operates as a predictive function on i.i.d. data, the notion of an agent is inextricable from the notion of dynamics. When an agent takes an action in its environment, that alters the state it will observe as its input on the following timestep. This cycle may continue indefinitely, or until some time limit or termination condition is reached. Rather than predictive accuracy or error, there is a far richer concept of dynamic task performance, which is aggregated over time. In order to gain a holistic understanding of agent behaviour, one must consider the entirety of this dynamic interaction, but a policy model (tree-structured or otherwise) only provides insight into the agent as an input-output function at a single timestep.¹² Since in many environments, “*individual actions do not have enough impact on their own*” to determine long-term outcomes [60], such single-timestep analysis is inescapably partial.

In the *Broad-XAI* framework for agent interpretability [61], policy models represent the shallowest level of analysis, with higher levels requiring that models include intentional information related to the agent’s goals, objectives, and beliefs, as well as how its actions change over time. Rephrasing using the language of queries introduced in Section 2.4, we wish not only to answer queries of the form “in this circumstance, what does the agent do?”, but also queries related to intentional and temporal information, such as “how good (in terms of task performance) is this circumstance?” and “what happens next?”. The focus of the remainder of this chapter is on building tree-based models of black box agents that do help to answer these other queries, by expanding the set of summary statistics they contain and optimise for.

3.7 TripleTree: A Multiattribute Tree Abstraction

We now present TRIPLETREE, our second tree-based model, which was conceived to address the shortcomings of policy-only tree models identified above. We begin with the premise that the lack of versatility of these models is a product of how observational data are stored in the tree, and of the algorithm used to grow it, rather than an inherent limitation of the tree-based abstraction paradigm itself. Far richer and more expressive tree models can be constructed by making fuller use of the available observational data.

TRIPLETREE is specifically tailored for MDP environments, where agent performance is measured by a reward function (see Section 1.2). In addition to summarising the actions taken by the agent in different subsets of the state space, the model also approximates its state value

¹²More accurately, it provides no insight when examined in isolation. As we have seen, a policy model can aid understanding of dynamic behaviour if paired with a simulator or recorded agent trajectory, but its accuracy is liable to increasingly diverge over longer time horizons.

function (discounted future reward), as well as the expected change in each state feature between successive timesteps. It is learnt using a novel variant of the CART algorithm, with a hybrid impurity measure that trades off the accuracy of all three summaries. After describing the model and exploring how the impurity tradeoff is managed by a weighting hyperparameter, we demonstrate how TRIPLETREE facilitates practical understanding of black box agents through prediction, visualisation, rule-based explanation and generation of hypothetical trajectories. As such, it provides a versatile tool for answering many meaningful queries about agent behaviour.

3.7.1 Data Preparation and Model Structure

As before, we adopt the perspective of a passive spectator of a black box agent executing a stationary policy in its environment, with the ability to observe environment states and agent actions over time. We assume that the environment is an MDP, meaning that each timestep is also associated with an observable real-valued reward, and is potentially episodic, meaning that it may occasionally terminate and reset at a new initial state. Our observations thus enable us to assemble a chronologically ordered dataset of N 4-tuples, $\mathcal{D} = \{(s_1, a_1, r_1, te_1), \dots, (s_N, a_N, r_N, te_N)\}$. For each $t \in \{1, \dots, N\}$, $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$ and $r_t \in \mathbb{R}$ are the state, action and reward from one timestep, uniquely indexed by t , and $te_t \in \{true, false\}$ is a Boolean indicator of whether that timestep terminates an episode. We further assume that a vector of interpretable state features $\phi(\mathcal{S}) = \mathbb{R}^D$ is readily available, or has been generated or learnt by some prior method.

We will make this assumption for all remaining methods presented in this thesis, but reiterate that the problem of interpretable feature generation arises frequently in practice. The operator-based feature generation method from Section 3.2.2 is compatible with all of our tree models.

Before constructing the TRIPLETREE model, we preprocess the dataset \mathcal{D} as follows. Firstly, a Monte Carlo value estimate is computed for each observation using the subsequent rewards:

$$(3.20) \quad v_t = \sum_{k=t}^{\text{next_te}(t)} \gamma^{k-t} r_k, \quad \text{where} \quad \text{next_te}(t) = \min\{k \in \{t, \dots, N\} : te_k = \text{true}\},$$

and $\gamma \in [0, 1]$ is a discount factor.¹³ This imbues each observation with richer information about the agent’s objective: not just the immediate reward, but the discounted sum of its future reward thereafter, which is the quantity that agents in MDPs work to maximise in expectation. Secondly, we wish to include information about the dynamics of the environment state. This can be captured by the time derivatives of state features, which in discrete-time systems are equal to the change between successive timesteps. For each observation $t \in \mathcal{D}$, this is defined as:

$$(3.21) \quad \Delta_t = \begin{cases} \phi(s_t) - \phi(s_{t-1}) & \text{if } te_t = \text{true}, \\ \phi(s_{t+1}) - \phi(s_t) & \text{otherwise,} \end{cases}$$

i.e. terminal observations use the change since the previous timestep, and all others use the change to the next one.¹⁴ Let $\mathcal{D}^+ = \{(s_1, a_1, v_1, \Delta_1, te_1), \dots, (s_N, a_N, v_N, \Delta_N, te_N)\}$ denote

¹³ γ is a free parameter, but if the agent uses RL it is sensible to match its internal γ value.

¹⁴We assume that all episodes have at least two timesteps; otherwise the derivatives would be undefined.

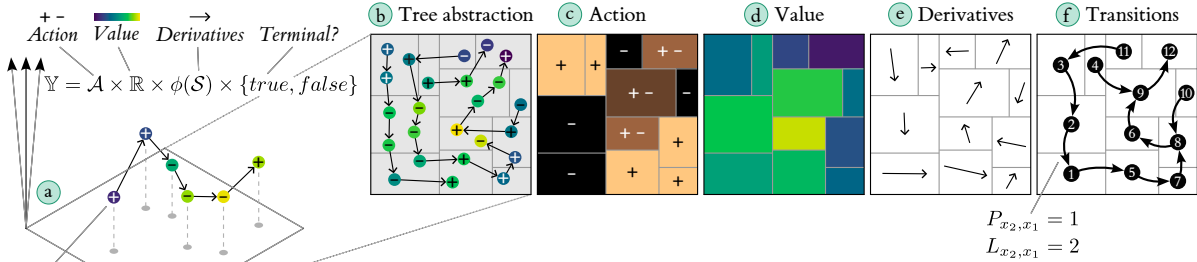


Figure 3.11: Conceptual space representation of TRIPLETREE.

the preprocessed dataset, containing value estimates and state feature derivatives in place of instantaneous rewards. Figure 3.11 (a) depicts this dataset in a conceptual space visualisation. Each observation is associated with a state feature vector in $\phi(\mathcal{S})$, and an action (+ or -), value estimate (purple-yellow colour scale), derivative vector (arrow) and terminal indicator as its attributes \mathbb{Y} . Note how the derivative vectors can be viewed as connecting each episode of observations into an unbroken trajectory in $\phi(\mathcal{S})$.

Following the general narrative of Chapter 2, our approach to understanding this dataset is to construct a tree-structured abstraction of $\phi(\mathcal{S})$, such as the one in Figure 3.11 (b), that is *efficient* in the sense that it has a small number of leaves, but nonetheless enables us to accurately answer queries about the attributes \mathbb{Y} using leaf-level summary statistics. If we were solely interested in constructing a policy model, as in the first half of this chapter, the aim would be to construct leaves containing observations with similar actions, which are well-summarised by their mean or modal value (represented visually in Figure 3.11 (c)). Alternatively, queries about the agent’s value function would be best answered by leaves with low internal variance according to this metric (e.g. (d)). Thirdly, we might seek a compact representation of the environment’s dynamics by finding leaves within which the state feature derivative vectors are similar; the mean vector in each leaf can be visualised as a *flow field* as shown in (e).¹⁵ Our key assertion in this work is that there is no need to choose between these three types of query, and that a powerful and versatile model results from identifying subsets of $\phi(\mathcal{S})$ that are similar from all three perspectives. The tree abstraction in Figure 3.11 is an example of this.

Our approach is thus an extension of the established technique of policy modelling. Alongside a mean or modal action, each leaf is associated with two additional summary statistics: an average value (the mean from the leaf’s constituent observations), and an average derivative vector (the elementwise mean), hence the name TRIPLETREE. The tree is grown using a hybrid of three corresponding impurity measures. By modifying a weight vector $\theta \in \mathbb{R}_+^3$, which sets the influence of the three measures, it is possible to smoothly trade off between three types of interpretable abstraction:

¹⁵Flow field plots are most meaningful for environments whose dynamics in $\phi(\mathcal{S})$ are temporally smooth, such as physical systems with small timestep intervals. This is the case for both environments used in our experiments.

- $\theta = [1, 0, 0]$: A conventional policy model, as described in the first half of this chapter.
- $\theta = [0, 1, 0]$: A model of the agent's value function.
- $\theta = [0, 0, 1]$: A model of the environment state dynamics, given the agent's policy.

Any other weighting gives a hybrid of the three, allowing for multiattribute analysis.

We complete the model by calculating transition probabilities between leaves: the probability that having been observed at a state in one leaf, the agent will move to another leaf next. These effectively define a probabilistic finite state machine approximation of the agent-environment dynamics, and can be estimated by harnessing the temporal ordering of \mathcal{D}^+ . This allows us to analyse more long-term dynamics than those captured by the state derivatives. Let $\text{leaf}(t) = x \in \mathcal{X} : \phi(s_t) \in x$ be the unique leaf containing the feature vector for observation t . For each observation, we find the first future observation that belongs to a different leaf,

$$(3.22) \quad \text{next_diff}(t) = \min\{k \in \{t + 1, \dots, N\} : \text{leaf}(k) \neq \text{leaf}(t)\},$$

then obtain the identity of this next leaf,

$$(3.23) \quad \text{next_leaf}(t) = \begin{cases} \emptyset & \text{if } \text{next_te}(t) < \text{next_diff}(t), \\ \text{leaf}(\text{next_diff}(t)) & \text{otherwise,} \end{cases}$$

where the special case for $\text{next_te}(t) < \text{next_diff}(t)$ handles episode termination. Next, for each $x \in \mathcal{X}$, we identify the timesteps whose predecessors are not themselves in x , i.e. the first in each sequence of successive observations that belong to that leaf:

$$(3.24) \quad \text{seq_starts}(x) = \{t \in \{1, \dots, N\} : \text{leaf}(t) = x \wedge (t = 0 \vee \text{te}_{t-1} = \text{true} \vee \text{leaf}(t-1) \neq x)\}.$$

Transition probabilities are estimated at the level of sequences, rather than individual timesteps, to avoid a double-counting effect. Overloading notation, we then find the subset of these timesteps whose successor sequences end with a transition to each other leaf (or termination \emptyset):

$$(3.25) \quad \text{seq_starts}(x, x') = \{t \in \text{seq_starts}(x) : \text{next_leaf}(t) = x'\}.$$

Finally, we compute the empirical probability that any given sequence in x ends in a transition to x' , and the mean length of such a sequence:

$$(3.26) \quad P_{x,x'} = \frac{|\text{seq_starts}(x, x')|}{|\text{seq_starts}(x)|}; \quad L_{x,x'} = \sum_{t \in \text{seq_starts}(x, x')} \frac{\min\{\text{next_te}(t), \text{next_diff}(t)\} - t}{|\text{seq_starts}(x, x')|}.$$

Note that in episodic environments, $P_{x,\emptyset}$ and $L_{x,\emptyset}$ are well-defined and meaningful; they are derived from sequences for which the episode terminates before a transition to another leaf. Transition probabilities and mean sequence lengths are stored at their respective source leaves (x here) as further summary statistics alongside the mean/modal actions, values and derivatives. Figure 3.11 (f) shows how this transition information can be represented as a directed graph over the leaves. In this example, the transition probability from leaf 2 to leaf 1 is 1 (i.e. leaf 2 is always followed by leaf 1) and the mean sequence length for this transition is 2.

3.7.2 Learning Algorithm

TRIPLETREE models are learnt by extending the CART algorithm outlined in Section 3.2.4. This extended algorithm trades off the accuracy of the action, value and derivative summary statistics by encouraging leaves to have low variability in all three attributes across their contained observations. To achieve this, we compute three measures of the quality of a candidate split of leaf x at threshold c along feature d :

- **Action quality** $Q^A(x, d, c)$: defined exactly as in Equation 3.10.
- **Value quality** $Q^V(x, d, c)$: defined equivalently, but using the variance in Monte Carlo value estimates as the impurity measure:

$$(3.27) \quad I^V(\mathcal{D}_x^+) = \mathbb{E}_{(s,a,v,\Delta,te) \in \mathcal{D}_x^+} \mathbb{E}_{(s',a',v',\Delta',te') \in \mathcal{D}_x^+} [(v - v')^2],$$

where \mathcal{D}_x^+ is the subset of observations that belong to leaf x , as per Equation 3.2.

- **Derivative quality** $Q^\Delta(x, d, c)$: defined equivalently, but using an impurity measure I^Δ that sums the variance in derivatives across all state features:

$$(3.28) \quad I^\Delta(\mathcal{D}_x^+) = \mathbb{E}_{(s,a,v,\Delta,te) \in \mathcal{D}_x^+} \mathbb{E}_{(s',a',v',\Delta',te') \in \mathcal{D}_x^+} [\mathbf{z}^\top (\Delta - \Delta')^2].$$

$\mathbf{z} \in \mathbb{R}^D$ is a vector of normalisation factors for all derivatives $d \in \{1, \dots, D\}$, which are defined as the reciprocals of their standard deviations across \mathcal{D}^+ . Normalisation prevents features with large derivative magnitudes from dominating the impurity calculation. This weighted-sum-of-variances impurity measure is similar to those used in prior work on multivariate regression trees [63, 141].

Q^A , Q^V and Q^Δ are then aggregated into a hybrid measure of split quality Q^* . Having experimented with alternative methods in various environments, we find that a linear combination provides a good compromise of simplicity, robustness and flexibility:

$$(3.29) \quad Q^*(x, d, c) = \left[\frac{Q^A(x, d, c)}{I^A(\mathcal{D}^+)}, \frac{Q^V(x, d, c)}{I^V(\mathcal{D}^+)}, \frac{Q^\Delta(x, d, c)}{I^\Delta(\mathcal{D}^+)} \right]^\top \theta.$$

$\theta \in \mathbb{R}_+^3$ is a weight vector, which trades off accurate modelling of the policy, value function and derivatives. Each quality term is normalised by the respective impurity across the entire dataset (i.e. at the root of the tree). This brings the three measures onto equivalent scales. As in CART, the d, c pair that greedily maximises the split quality is chosen, thereby splitting x into two new leaves.

The description of CART in Section 3.2.4 follows a depth-first growth strategy starting at the root, and continuing until all leaves have zero impurity. Since the tree now predicts multiple continuous variables, zero impurity is only likely to be achieved when every observation is

contained in its own leaf (i.e. $|\mathcal{X}| \approx |\mathcal{D}^+|$). This would lead to both excessive growth runtime and a uselessly large and overfitted tree. For this reason, for TRIPLETREE the criterion for terminating tree growth is a limit on the number of leaves $|\mathcal{X}|$, paired with a simple best-first strategy for selecting which leaf to split at each stage of tree growth (if the depth-first strategy were retained, it would only ever grow a single branch). Again taking a hybrid view of impurity, the best current leaf to split, x_{best} , can be identified as follows:

$$(3.30) \quad x_{\text{best}} = \operatorname{argmax}_{x \in \mathcal{X}} |\mathcal{D}_x^+| \left[\frac{I^A(x, d, c)}{I^A(\mathcal{D}^+)}, \frac{I^V(x, d, c)}{I^V(\mathcal{D}^+)}, \frac{I^\Delta(x, d, c)}{I^\Delta(\mathcal{D}^+)} \right]^\top \theta,$$

where θ is the same as in Equation (3.29). This approach prioritises splitting leaves with high total impurity, weighted by their number of observations. Although we do not implement pruning in our experiments, this could be done with a small modification of the MCCP algorithm.

In summary, the key features of TRIPLETREE are:

- Acceptance of the ordered 4-tuple observation dataset \mathcal{D} , calculation of value and state derivatives for each sample, and storage of predicted values of these variables at each leaf.
- A hybrid measure of split quality Q^* , mediated by a weight vector θ , which trades off the tree’s abilities to predict the target agent’s action, value and state derivatives.
- Calculation of $P_{x,x'}$ and $L_{x,x'}$ to encode information about temporal dynamics in terms of leaf-to-leaf transitions.
- A best-first growth strategy.

3.7.3 Related Work

The TRIPLETREE model can be classified as a multi-task tree [85], which is grown and evaluated for multiple predictive tasks. Various aspects of this proposal have some precedence in the literature, but they have never been combined into a unified model. For example, [157] use the target agent’s value function to locally weight the leaf impurity measure during policy modelling, in order to prioritise more critical states, while [132] and [163] use trees to approximate the value function itself as a route to interpretability. The latter model also keeps track of transition probabilities between tree leaves, similar to our approach. In [133], a tree is grown to minimise the impurity of environment state derivatives as part of a model-based RL framework, and in [141] a tree is optimised for sequential prediction by jointly minimising loss on consecutive timesteps. Other work has looked at approximating a recurrent neural network policy as a finite state transition model for visualisation and analysis [150]. We know of one work that considers a hybrid action- and value-based tree impurity measure [222], but the idea is tangential to the main topic of the paper and its implications are left unconsidered. We are unaware of any prior work that jointly represents the policy, value function and temporal dynamics of an agent in one tree model, or considers the benefits of doing so for interpretability.

3.8 Quantitative Evaluation

We initially validate TRIPLETREE in a simple continuous environment with two state features and two discrete actions. This can be interpreted as a straight road, down which a vehicle agent can drive in either direction. The state features $\phi(\mathcal{S})$ are position $pos \in [0, 3]$ (increasing left-to-right) and signed velocity $vel \in [-0.1, +0.1]$, and the agent’s action is a small positive or negative acceleration $acc \in \{-0.001, +0.001\}$ (we use $-$ and $+$ as shorthand symbols for these actions). Walls lie at the left and right ends of the road. A collision with either yields a reward of R_{left} and R_{right} respectively and instantly terminates the simulation episode. The agent also receives reward in each non-terminal state in proportion to its absolute speed: $R_{\text{speed}} \times |vel|$. Figure 3.12 summarises this information.

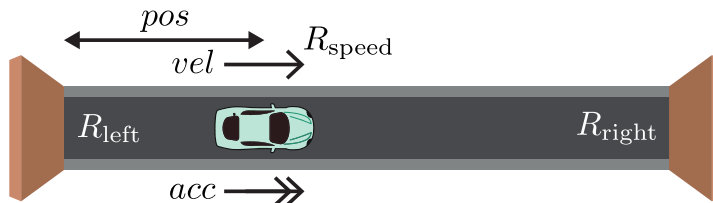


Figure 3.12: The 2-dimensional road environment.

For a given R_{left} , R_{right} , R_{speed} , discount factor γ (we use $\gamma = 0.99$), and suitable discretisation of the state space (we use a 30×30 grid) an optimal policy can be found by dynamic programming (DP). The DP policies for four reward function variants are used as the target agents in our experiments. For each, we create a dataset \mathcal{D} with 10^4 samples by running randomly initialised episodes of 100 timesteps, then calculate Monte Carlo value estimates and state derivatives to yield the preprocessed dataset \mathcal{D}^+ . We then grow a TRIPLETREE of up to 200 leaves on each of the four datasets, with various weightings θ that prioritise only action, value or derivative quality, or prioritise all three equally. Figure 3.13 plots three predictive losses as a function of leaf count for these trees:

- **Action loss:** Proportion of observations whose action does not equal the leaf-level mode.
- **Value loss:** Root mean square (RMS) error between value estimates and leaf-level means.
- **Derivative loss:** RMS errors between derivatives and leaf-level means, scaled by normalisation vector \mathbf{z} and summed.

Naturally, different trees result when different θ vectors are used, and in all cases, the lowest loss of each type is obtained by exclusively using the corresponding split quality measure. Crucially, however, using an equal weighting ($\theta = [1/3, 1/3, 1/3]$; black curves) offers a strong compromise between the three modes of prediction. For action and derivatives, equal weighting converges slower than exclusive weighting, but to virtually the same asymptotic loss, with the greatest disparity for trees with around 50-100 leaves. For value, the gap is more significant. This

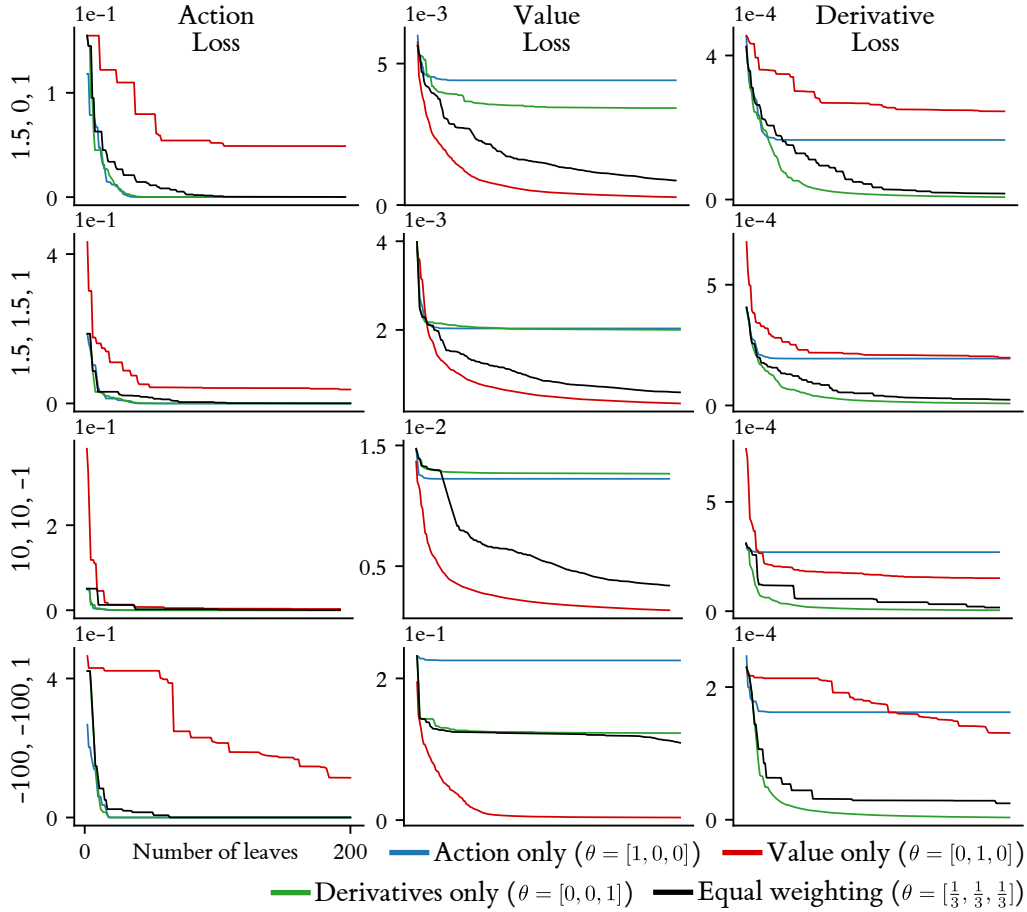


Figure 3.13: Prediction losses for four variants of the road environment, with R_{left} , R_{right} , R_{speed} as stated in the left-hand labels.

indicates that there tend to be regions of the feature space in which value varies significantly but the agent’s action and state derivatives do not (or vice versa), thereby creating a conflict as to which splits are worth making. This phenomenon is most pronounced for the policy on the bottom row. Another notable trend is that splitting on derivatives alone does very well in terms of action loss. This makes perfect sense upon noting that the agent’s action (acc) is exactly the time derivative of one of the state features (vel) in this environment. Of course, such a relationship should not be expected to hold in general.

This analysis begs the question: what is the best choice of θ for this environment? As in any multi-objective optimisation problem, there is no universal answer to this question, and the most appropriate choice of θ ultimately depends on the intended application, but if general versatility is important then it is reasonable to consider minimising the worst of the three loss types. To investigate worst-case loss behaviour, we grow a `TRIPLETREE` for each of 21 equally-spaced weighting vectors θ . For each, we calculate which loss is highest as a ratio of the loss of a tree with just one leaf, whose summary statistics are just the dataset averages. The results are shown on simplex plots in Figure 3.14.

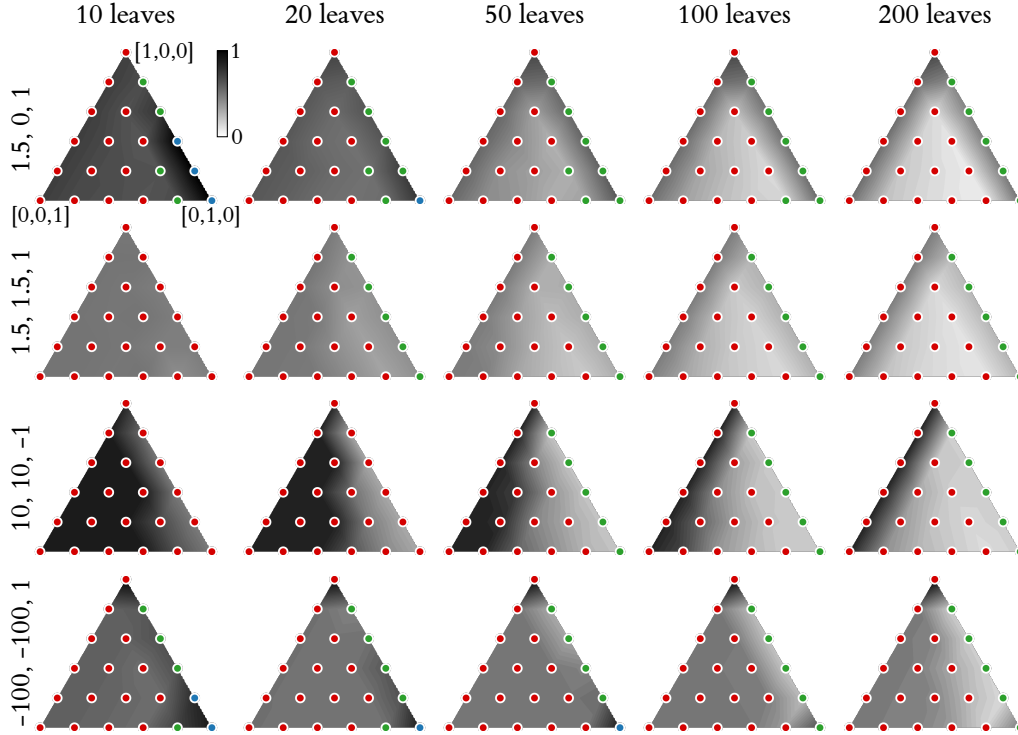


Figure 3.14: Analysis of worst-case loss across the space of weight vectors for trees of various sizes. Coloured dots show the identity of the worst loss (blue: action, red: value, green: derivatives) at the 21 weightings tested. Heatmaps show the magnitude of the worst loss as a ratio of the loss from a one-leaf tree, linearly interpolated between test locations.

The prevalence of red dots throughout the range of tree sizes shows that value prediction is weakest across most of the space of weightings. A notable exception is the right-hand edge of the simplex, where derivative weight is 0 and that loss is accordingly the worst. The greyscale heatmaps show the magnitude of the worst loss ratio (interpolated using the `LinearTriInterpolator` function in the `matplotlib` Python library), which as a general rule is lower towards the centre of the simplex, but also towards the right-hand side, where the value weight is higher. These results point to the conclusion that value should be up-weighted in the hybrid quality metric if minimising the worst-case loss ratio is important. From looking at the various heatmaps, we suggest that $\theta = [0.2, 0.6, 0.2]$ is a good compromise. This weighting is used for all remaining experiments in the road environment.

3.9 Model Interpretation and Explanation

This section demonstrates some of the ways in which `TRIPLETREE` can be used to gain both global and local understanding of agent behaviour. While the equivalent discussion in Section 3.5 was largely focused on the diagrammatic, rule-based representation of trees, here we emphasise their alternative representation as geometric structures that partition their feature spaces, and explore how this provides a natural mechanism for visualisation.

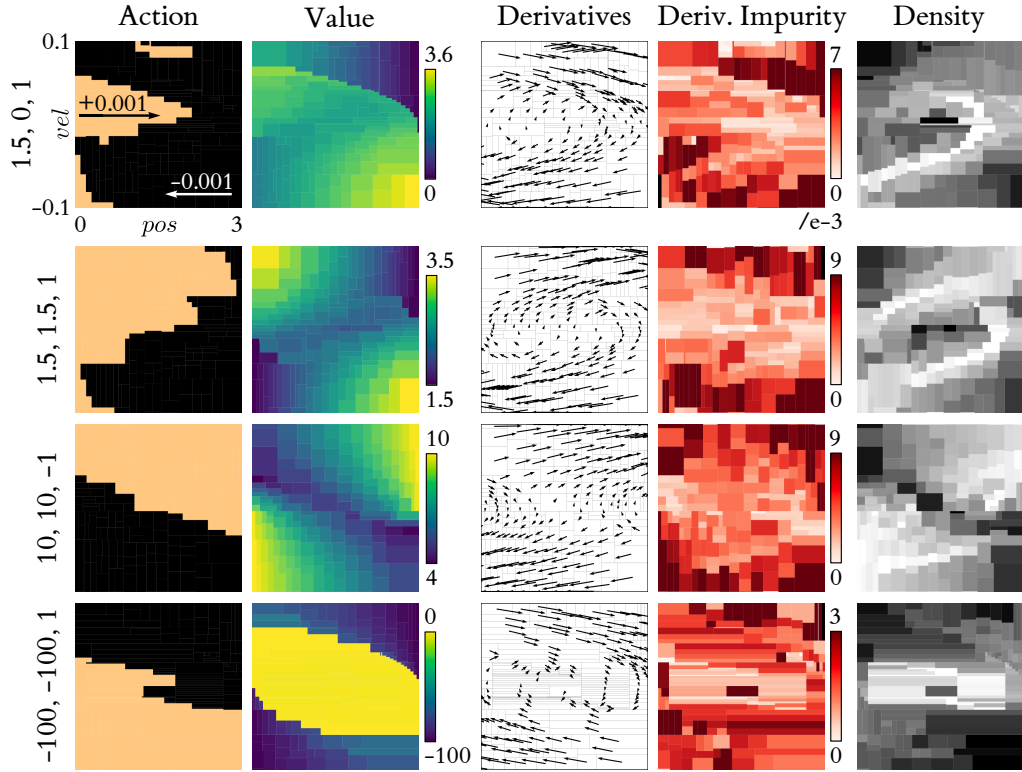


Figure 3.15: Five types of visualisation using TRIPLETREE.

3.9.1 Multiattribute Visualisation in Feature Space

Recall once again that in an axis-aligned tree, each leaf is associated with a hyperrectangle in the D -dimensional feature space (here derived from the environment’s state space, $\phi(\mathcal{S})$), whose boundaries correspond to the splits made at its ancestors. For $D = 2$, hyperrectangles reduce to rectangles, which can be directly shown on axes corresponding to $\phi(\mathcal{S})$ itself (we return to the $D > 2$ case in Section 3.10.1). Each leaf can be coloured according to a summary statistic including, but not limited to, its action, value or derivative prediction.

Figure 3.15 demonstrates the rich information conveyed by such visualisations in the road environment. Each row of plots represents a different policy, and is generated from a single tree with 200 leaves, grown using the weighting $\theta = [0.2, 0.6, 0.2]$. In the first column, leaves are coloured by their modal action, revealing the global geometry of the four optimal DP policies. The decision boundaries between positive and negative acceleration have varying complexity; interesting features include the isolated ‘island’ of positive acceleration in the top policy, which occurs when a crash with the right wall is unavoidable but positive R_{speed} can be obtained by accelerating in the meantime, and the z-shaped geometry of the bottom policy, which causes the agent to oscillate around the centre of the road to avoid hitting either wall (reward = -100).

In the second column, colours denote the predicted value, which intuitively reflects the differing reward structures. In general, low value corresponds to an imminent crash into a

low-reward wall. Value is high when the agent approaches a high-reward wall and/or has plenty of room to accumulate positive R_{speed} . For the bottom policy, value is high within a boundary of stability for the oscillatory motion, and low elsewhere.

The plots of predicted derivatives in the third column differ from the others. Since this is a vector quantity, it can be depicted as a flow field with an arrow for each leaf, whose direction and magnitude reflect the mean change in state between successive timesteps. The system changes more rapidly at high speeds, hence the longer arrows in these areas. Flow fields provide an excellent high-level overview of system dynamics, particularly the locations of directional changes, cycles and regions of constancy.

The fourth column colours leaves by derivative impurity, showing where in $\phi(\mathcal{S})$ the model’s derivative predictions have low and high uncertainty. In general, impurity is higher in spatially larger leaves, which is intuitive as these can contain larger portions of curved trajectories. There is especially high impurity at the centre of the oscillatory motion of the bottom policy, where there is a sharp discontinuity in the agent’s direction of motion. Equivalent plots can be created for action and value impurity.

The final colouring statistic is sample density, computed by dividing the number of observations contained in each leaf by its ‘volume’ in $\phi(\mathcal{S})$: the product of boundary lengths, normalised by each feature’s range across \mathcal{D}^+ . This reveals where the policies spend the most time: in narrow arcs for the top two, and a tight central patch for the bottom one.

3.9.2 Factual, Counterfactual and Temporal Explanation

The methods for rule-based explanation presented in Sections 3.5.2 to 3.5.4 can be extended to work with TRIPLETREE, and complemented with visualisations, as shown in Figure 3.16. Here, (a) shows a zoomed-in portion of the action visualisation for one of the 200-leaf trees from the previous section. Suppose that the state feature vector for an observation at time t is $\mathbf{f}^t = \phi(s_t) = [1.26, 0.037]$. This vector belongs to a leaf with positive acceleration (+) as its modal action. The factual explanation, derived from the boundaries of this leaf, is

“Action is + because $1.10 \leq pos < 1.32$ and $0.021 \leq vel < 0.045$ ”,

and the minimal counterfactual for the foil action of negative acceleration (-) is \mathbf{f}' , i.e.

“Action would be - if vel increased to ≥ 0.045 ”.

The same TRIPLETREE model allows us to similarly explain value predictions. Since value is continuous, a counterfactual query could ask why value is less than or greater than a threshold, rather than defining a precise numerical foil, which is unlikely to be satisfied by any of the leaves. In Figure 3.16 (b), the foil condition $v \leq 0.3$ leads to the following minimal counterfactual for the predicted value at $\mathbf{f}^t = \phi(s_t) = [1.60, -0.0148]$:

“Value would be ≤ 0.3 if pos increased to ≥ 2.64 and vel increased to ≥ 0.024 ”.

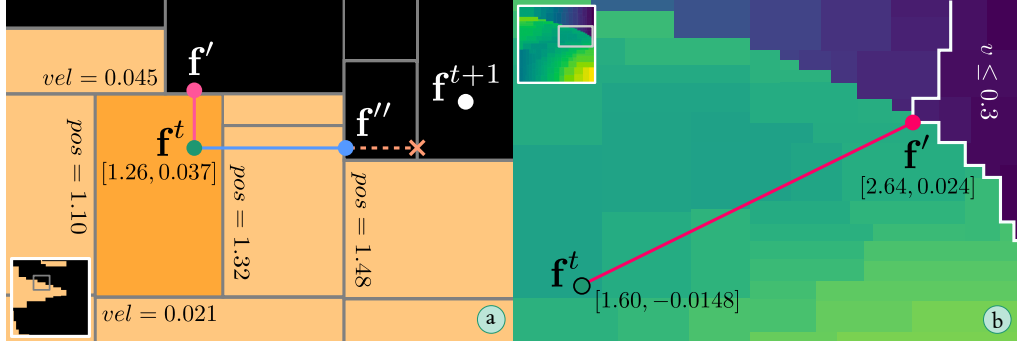


Figure 3.16: Various kinds of rule-based explanation for action and value.

In general, a foil can be defined as any function that returns *true* or *false* given a leaf $x \in \mathcal{X}$. Let $X_{\text{foil}} = \{x \in \mathcal{X} : \text{foil}(x) = \text{true}\}$ denote the subset of leaves that satisfy a given foil. This notation unifies the generation of counterfactuals with respect to action, value, and even derivatives (e.g. the foil may test for a certain component of the mean derivative vector exceeding a threshold), all of which can be produced using a single TRIPLETREE model.

Furthermore, the general form of the temporal explanation problem introduced in Section 3.5.4 arises when for two consecutive timesteps t and $t + 1$, the state feature vectors \mathbf{f}^t and \mathbf{f}^{t+1} belong to different leaves $x^t \neq x^{t+1}$, such that for some foil condition, $x^t \notin X_{\text{foil}}$ and $x^{t+1} \in X_{\text{foil}}$. Returning to Figure 3.16 (a), we find a simple example of this: the state moves from a leaf with a modal action of $+$ to one with a modal action of $-$. Following the method of Section 3.5.4, a textual explanation of this change could be generated by identifying the closest point to \mathbf{f}^t within the leaf containing \mathbf{f}^{t+1} , indicated by the cross. However, this feature space visualisation reveals a weakness of that approach: the explanation would not be minimal, because the leaf containing \mathbf{f}^{t+1} is surrounded by other leaves that also have the foil action. Instead, it would seem more natural to use \mathbf{f}' as the basis of the explanation, i.e.

“Action changed from $+$ to $-$ because pos increased to ≥ 1.48 ”.

Since this differs from the minimal counterfactual when not conditioning on \mathbf{f}^{t+1} (i.e. \mathbf{f}'), we are led to suspect that neither the naïve approach described in Section 3.5.4, nor the standard counterfactual method from Section 3.5.3, is the best way to do temporal explanation.

To further investigate and formalise this intuition, consider Figure 3.17 (a), which echoes Figure 3.10 presented earlier. Here $X_{\text{foil}} = \{x_1, \dots, x_8\}$, but every one of $\mathbf{f}^{(t \rightarrow x_1)}, \dots, \mathbf{f}^{(t \rightarrow x_8)}$ turns out to be a suboptimal counterfactual. Since \mathbf{f}^{t+1} lies in x_4 , the naïve approach would be to use $\mathbf{f}^{(t \rightarrow x_4)}$, but this does not achieve minimality because there are foil-satisfying points (e.g. in x_5) that are unambiguously (i.e. under any p -norm) closer to \mathbf{f}^t . According to the standard method of counterfactual generation from Section 3.5.3, $\mathbf{f}^{(t \rightarrow x_1)}$ would be minimal, but it implies moving in the opposite direction from \mathbf{f}^{t+1} so does not lie on any plausible motion path between the two feature vectors. $\mathbf{f}^{(t \rightarrow x_8)}$ is at least closer to \mathbf{f}^{t+1} than \mathbf{f}^t is, but there is not an unbroken path of foil leaves connecting it to \mathbf{f}^{t+1} , so it does not represent a sufficient condition for the

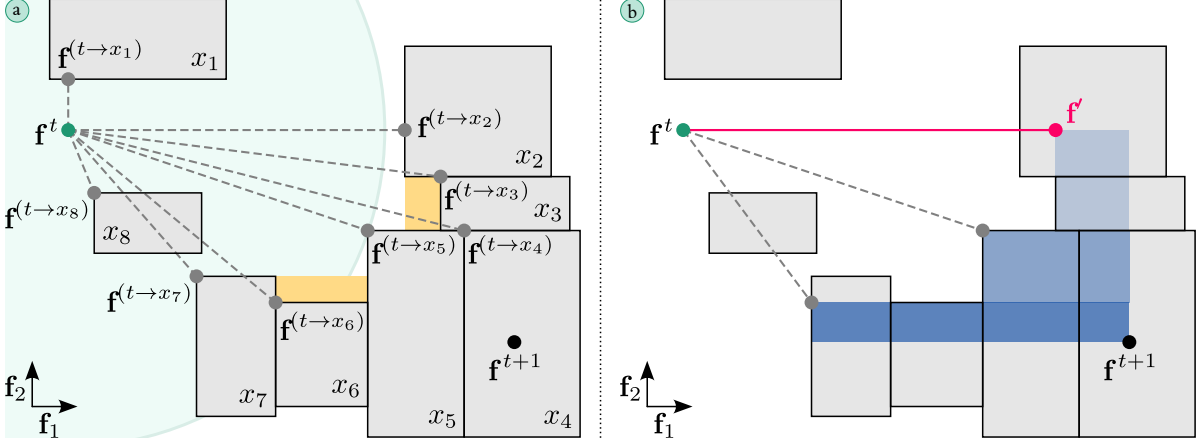


Figure 3.17: Illustrating the inadequacy of previous methods for counterfactual explanation in the temporal context, and the proposed MBB-based approach.

foil to be realised. More subtly, the same is true for both $\mathbf{f}^{(t \rightarrow x_2)}$ and $\mathbf{f}^{(t \rightarrow x_7)}$; the yellow shading highlights regions of $\phi(\mathcal{S})$ below and to the right of these locations, but which are not inside any $x \in X_{\text{foil}}$. The existence of these regions means that a textual explanation such as

“Foil became satisfied between t and $t + 1$ because feature 1 increased to $\geq \mathbf{f}_1^{(t \rightarrow x_2)}$,”

would be misleading, since it is possible to satisfy the stated condition in a location between \mathbf{f}^t and \mathbf{f}^{t+1} while not satisfying the foil. All three of $\mathbf{f}^{(t \rightarrow x_3)}$, $\mathbf{f}^{(t \rightarrow x_5)}$ and $\mathbf{f}^{(t \rightarrow x_6)}$ are acceptable according to this misleadingness criterion, although $\mathbf{f}^{(t \rightarrow x_6)}$ can be ruled out for the same reason as $\mathbf{f}^{(t \rightarrow x_4)}$. Of the two remaining options, $\mathbf{f}^{(t \rightarrow x_5)}$ is closer to \mathbf{f}^t by 2-norm (faint green circle), so it may initially appear that by elimination, this is the best possible counterfactual.

However, an even better solution is possible. We propose that the best counterfactual to use for temporal explanation, \mathbf{f}' , is the location inside $\bigcup_{x \in X_{\text{foil}}} x$ which is minimal from \mathbf{f}^t , subject to the constraint that the *minimum axis-aligned bounding box* (MBB) of \mathbf{f}' and \mathbf{f}^{t+1} only intersects leaves in X_{foil} . This constraint ensures that the generated counterfactual explanation is never misleading in the sense outlined above. Figure 3.17 (b) shows the optimal counterfactual for the example shown, for which the textual explanation is

“Foil became satisfied between t and $t + 1$ because feature 1 increased to $\geq \mathbf{f}'_1$ ”.

This counterfactual is preferred to the other two options shown in grey (one of which is equivalent to the previous best guess of $\mathbf{f}^{(t \rightarrow x_5)}$) due to the strict priority of the 0-norm over the 2-norm. Notice how this MBB-based result is nontrivial: \mathbf{f}' does not lie on the boundary of its containing leaf (x_2), so would never even be considered by the standard method from Section 3.5.3.

Finding such an \mathbf{f}' in practice is also nontrivial, and we have currently only derived a method for 2D feature spaces, outlined in Algorithm 1. This algorithm returns X_{MBB} , a set of rectangles tiling the region of $\phi(\mathcal{S})$ between \mathbf{f}^t and \mathbf{f}^{t+1} that satisfies the MBB constraint. The minimal counterfactual equations from Section 3.5.3 are then applied to X_{MBB} instead of the

underlying foil leaves X_{foil} ,¹⁶ thereby obtaining the optimal counterfactual. In the example in Figure 3.17 (b), X_{MBB} is composed of the three rectangles shown in blue, with dark-to-light shading according to the order in which they are generated by Algorithm 1.

Algorithm 1 Finding the region of $\phi(\mathcal{S})$ satisfying the MBB constraint (for $D = 2$ only)

Inputs: Feature vectors \mathbf{f}^t and \mathbf{f}^{t+1} , tree abstraction \mathcal{X} and foil leaves X_{foil}
Output: A set of rectangles X_{MBB} tiling the MBB-satisfying region

```

1: Initialise  $\mathbf{a} \leftarrow \mathbf{f}^{t+1}$ ,  $X_{\text{MBB}} \leftarrow \{\}$ 
2: while loop not yet broken do
3:    $\mathbf{b} \leftarrow \mathbf{a}$ 
4:    $\mathbf{b}_1 \leftarrow \text{EXPAND}(\mathbf{a}, \mathbf{b}, 1, 2)$  ▷ Expand rectangle along first feature axis
5:    $\mathbf{b}_2 \leftarrow \text{EXPAND}(\mathbf{a}, \mathbf{b}, 2, 1)$  ▷ Expand rectangle along second feature axis
6:   if  $\mathbf{b}_2 = \mathbf{a}_2$  then
7:     break ▷ Break if no expansion possible
8:   end if
9:    $X_{\text{MBB}} \leftarrow X_{\text{MBB}} \cup \{\text{RECTANGLE}(\mathbf{a}, \mathbf{b})\}$  ▷ Store rectangle
10:   $\mathbf{a}_2 \leftarrow \mathbf{b}_2$  ▷ Move  $\mathbf{a}$  along second feature axis
11: end while

12: function  $\text{EXPAND}(\mathbf{a}, \mathbf{b}, d, d')$  ▷ Given a rectangle with opposite corners at  $(\mathbf{a}, \mathbf{b})$ , move  $\mathbf{b}$  along feature axis  $d$  until the rectangle either meets a non-foil leaf  $x \in \mathcal{X} \setminus X_{\text{foil}}$  or extends as far as  $\mathbf{f}_d^t$ 
13:    $l \leftarrow \min\{\mathbf{a}_{d'}, \mathbf{b}_{d'}\}$ ,  $u \leftarrow \max\{\mathbf{a}_{d'}, \mathbf{b}_{d'}\}$  ▷ Lower/upper bounds of rectangle along  $d'$ 
14:    $X_{\text{-foil}} \leftarrow \{x \in \mathcal{X} \setminus X_{\text{foil}} : (\text{lo}(x, d') \leq u) \wedge (\text{hi}(x, d') \geq l)\}$  ▷ ‘Reachable’ non-foil leaves
15:   if  $\mathbf{f}_d^t > \mathbf{f}_d^{t+1}$  then ▷ Extend in the positive direction
16:     return  $\min(\{\text{lo}(x, d) : (x \in X_{\text{-foil}}) \wedge (\text{lo}(x, d) \geq \mathbf{a}_d)\} \cup \{\mathbf{f}_d^t\})$ 
17:   else ▷ Extend in the negative direction
18:     return  $\max(\{\text{hi}(x, d) : (x \in X_{\text{-foil}}) \wedge (\text{hi}(x, d) \leq \mathbf{a}_d)\} \cup \{\mathbf{f}_d^t\})$ 
19:   end if
20: end function
    
```

3.9.3 Hypothetical Trajectories

For each leaf $x \in \mathcal{X}$, the mean derivative vector and outbound transition probabilities provide complementary summaries of the agent’s movement through that subset of the state feature space. These can be combined to generate behavioural trajectories which may never have occurred in \mathcal{D}^+ , but are nonetheless realistic given the agent’s policy. Such a generative model could be useful for answering various kinds of hypothetical query about the agent’s expected behaviour before, after and between states of interest. This subsection considers the problem of finding such a trajectory between a given initial leaf x_0 and a goal leaf x_{goal} , thereby addressing queries of the form “starting in x_0 , is the agent likely to reach x_{goal} in future, and if so, how?”.

¹⁶The placeholder function `RECTANGLE` in Algorithm 1 creates a representation of each rectangle which can be fed into these equations in the same way as the leaves themselves.

As briefly mentioned in Section 3.7.1, the set of all transition probabilities can be represented as a directed graph over the leaves of the tree. Dijkstra’s graph search algorithm [72] can therefore be used to find a sequence of leaves $X_{0 \rightarrow \text{goal}} = (x_0, x_1, x_2, \dots, x_{\text{goal}})$ that the agent moves through with nonzero probability. We define the cost of each transition $x \rightarrow x'$ as the negative logarithm of the transition probability, $-\log P_{x,x'}$, and the cost of a full sequence as the sum of its constituent transitions. If finite-cost sequences exist between x_0 and x_{goal} , Dijkstra’s algorithm is guaranteed to find the lowest-cost (i.e. highest-probability) one first. If no such sequences exist, the algorithm returns a null result, which is itself potentially valuable information about the non-reachability of states (i.e. “the agent cannot reach x_{goal} from x_0 ”).

A non-null Dijkstra result indicates which leaves the agent is most likely to move through on a trajectory from x_0 to x_{goal} , but does not tell us *how* it moves through them. To generate a realistic trajectory through $X_{0 \rightarrow \text{goal}}$, we solve a constrained optimisation problem to build a piecewise linear path whose segments are well aligned with the leaves’ mean derivative vectors. Concretely, we initialise a path node $\mathbf{f}^i \in \phi(\mathcal{S})$ on the hyperrectangle boundary of each leaf $x_i \in X_{0 \rightarrow \text{goal}}$,¹⁷ then perform gradient descent on all node locations to minimise the squared angle between each path segment $\mathbf{f}^i \rightarrow \mathbf{f}^{i+1}$ and the corresponding leaf’s mean derivative vector, denoted by \mathbf{d}^i . When calculating angles, feature magnitudes are normalised by \mathbf{z} , the vector of inverse standard deviations across \mathcal{D}^+ . The unconstrained update to each \mathbf{f}^i is proportional to the partial derivative of the sum of squared angles for the segments before and after:¹⁸

$$(3.31) \quad \frac{\partial}{\partial \mathbf{f}^i} \left[\left(\cos^{-1} \frac{(\mathbf{f}_{\text{norm}}^i - \mathbf{f}_{\text{norm}}^{i-1})^\top \mathbf{d}_{\text{norm}}^{i-1}}{\|\mathbf{f}_{\text{norm}}^i - \mathbf{f}_{\text{norm}}^{i-1}\| \|\mathbf{d}_{\text{norm}}^{i-1}\|} \right)^2 + \left(\cos^{-1} \frac{(\mathbf{f}_{\text{norm}}^{i+1} - \mathbf{f}_{\text{norm}}^i)^\top \mathbf{d}_{\text{norm}}^i}{\|\mathbf{f}_{\text{norm}}^{i+1} - \mathbf{f}_{\text{norm}}^i\| \|\mathbf{d}_{\text{norm}}^i\|} \right)^2 \right],$$

where $\mathbf{f}_{\text{norm}}^i = \mathbf{f}^i \circ \mathbf{z}$ and $\mathbf{d}_{\text{norm}}^i = \mathbf{d}^i \circ \mathbf{z}$ (\circ is the Hadamard product). Instead of applying the update directly, we constrain each node to stay on its respective boundary, and always ‘visible’ from the previous node (i.e. it never moves to the far side of the boundary). We find that the optimisation usually converges to yield smooth and realistic trajectories, but is rather expensive and vulnerable to local minima. Further technical refinements would certainly be possible.

Figure 3.18 (a) - (d) depicts four trajectories generated by this Dijkstra-then-align method from the 200-leaf TRIPLETREES. Derivative arrows from nearby leaves indicate that the trajectories align well with the agent’s true motion in each region of the feature space, which is locally parabolic due to the underlying constant-acceleration dynamics. Notice that because both the initial and final path nodes are subject to the alignment optimisation, their locations indicate where specifically inside x_0 and x_{goal} the agent is most likely to begin and end its trajectory, given the local derivative information.

Further insight can be gained by combining hypothetical trajectories with other information contained in the TRIPLETREE model, such as leaf-level action and value predictions, as shown in Figure 3.18 (e) and (f). These indicate how we should expect the agent’s action and value

¹⁷Let $x_{k-1} = x_{\text{goal}}$, where $k = |X_{0 \rightarrow \text{goal}}|$.

¹⁸The first and second terms are excluded for the first and last path nodes respectively.

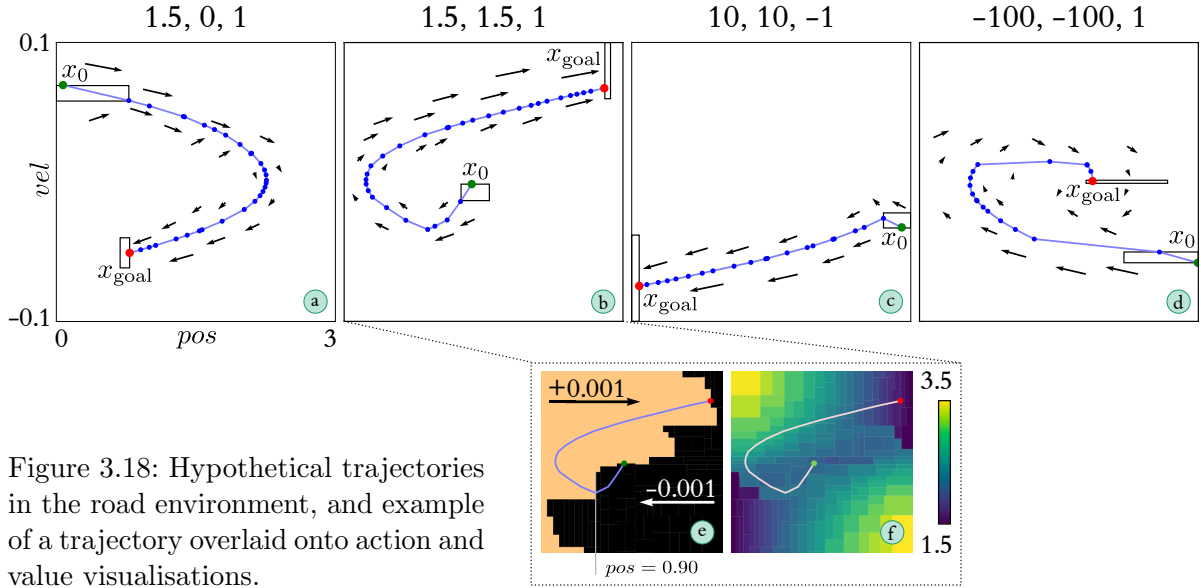


Figure 3.18: Hypothetical trajectories in the road environment, and example of a trajectory overlaid onto action and value visualisations.

function to change over time as it follows a trajectory from x_0 to x_{goal} . The action visualisation is especially informative, as it reveals that the agent is expected to change its action just once, when it crosses the threshold $pos = 0.9$. We can thus give a compact textual summary of the hypothetical trajectory:

“From x_0 , the agent reaches x_{goal} by taking action - until $pos = 0.9$, then switching to action +”.

This is a good example of the explanatory versatility of the TRIPLETREE model.

3.10 Experiments in a Higher-dimensional Environment

We now deploy TRIPLETREE in a more complex environment: LUNARLANDERCONTINUOUS-V2 from the *Gymnasium* library [86]. The objective is to guide an aerial craft to a gentle landing on a landing pad surrounded by uneven terrain. The state is represented by an 8-dimensional feature vector $\phi(s) = [pos_x, pos_y, vel_x, vel_y, \psi, vel_\psi, c_l, c_r]$, which are respectively the horizontal and vertical position and velocity, orientation, and angular velocity of the landing craft, and binary flags as to whether its left and right legs contact the ground. The action space $\mathcal{A} = [-1, 1]^2$ is bounded and 2D. The first component is the throttle for the lander’s main engine (-1 is off) and the second is a left-right side engine (0 is off). The reward function includes terms to promote smooth flight (by penalising high velocities and linear/angular accelerations), disincentivise fuel burn and reward leg-to-ground contact. There are also one-off rewards of $+100$ if the craft successfully lands on the pad, and -100 if it crashes or drifts out-of-bounds ($|pos_x| \geq 1$). Episodes terminate if a landing, crash or out-of-bounds event occurs, or after 400 timesteps.

The black box target policy in this environment is that of a soft actor-critic (SAC) [107] RL agent from *Baselines Zoo* [203], the highest-performing one on that repository. Using a

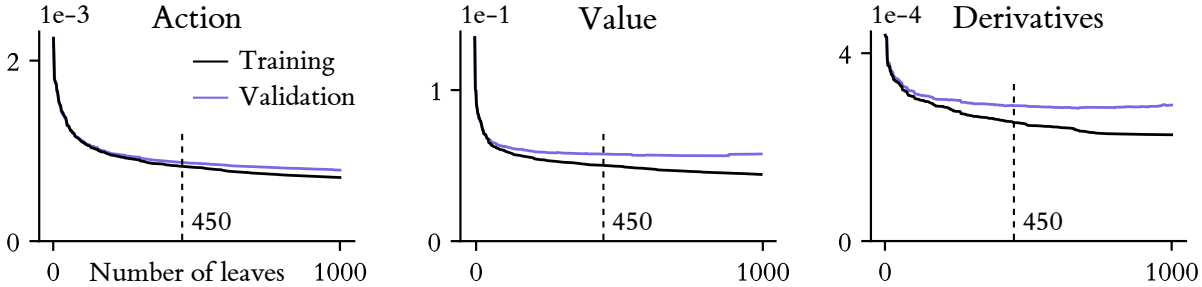


Figure 3.19: Training and validation losses in LunarLander.

dataset of 10^5 observations of this policy, we grow a `TRIPLETREE` of up to 1000 leaves with $\theta = [1, 1, 1]$.¹⁹ Figure 3.19 shows how the three losses vary during growth on both the training set and a validation set of the same size. In this more complex environment, the prediction problem is harder (particularly, it seems, for derivatives) and losses do not reduce to near zero, but as we shall see, the model still captures enough of the statistical properties of the system to deliver significant insight. Rather than implementing a pruning stage, we manually identify the point at which the validation losses plateau, leading us to select the 450-leaf tree for evaluation.

3.10.1 Hyperrectangle Projection for $D > 2$

Our analysis in the road environment benefited from the ability to visualise trees as rectangular partitions of the feature space, and colour each leaf according to one of its summary statistics. However, with an 8D feature space, it is nontrivial to create such visualisations; an assumption must be made about how to represent the high-dimensional partition on the 2D plane. Inspired by a prior taxonomy of geometric operations for transforming multidimensional “*space-time cubes*” into 2D data visualisations [16], we propose two ways forward: *projection* and *slicing*. In the former, leaf hyperrectangles are projected onto a plane defined by two chosen feature axes. Where multiple projections overlap, a marginal value for the colouring statistic is computed as a population-weighted average. This creates a *partial dependence plot* (PDP) of the statistic over the two features. PDPs are popular tools in the AI interpretability literature [180].

Suppose that the two features chosen for projection are d and d' , and we wish to create a visualisation with axis limits $[l, h]$ along d and $[l', h']$ along d' . Let \mathcal{B}^d denote the sorted sequence of hyperrectangle boundaries in the tree within the limits along d :

$$(3.32) \quad \mathcal{B}^d = \left(b \in \bigcup_{x \in \mathcal{X}} \{\max\{\text{lo}(x, d), l\}, \min\{\text{hi}(x, d), h\}\} \right), \text{ s.t. } \mathcal{B}_i^d > \mathcal{B}_{i-1}^d, \forall i \in \{2..|\mathcal{B}^d|\},$$

and $\mathcal{B}^{d'}$ denote the equivalent for d' . The planar region $[l, h] \times [l', h']$ can be tiled by a $(|\mathcal{B}^d| - 1) \times (|\mathcal{B}^{d'}| - 1)$ grid of rectangles, each of which is constructed from pairs of consecutive boundaries from \mathcal{B}^d and $\mathcal{B}^{d'}$. Steps (a) and (b) in Figure 3.20 illustrate this reasoning.

¹⁹Since the action space has two continuous dimensions on the same $[-1, 1]$ scale, we use Euclidean distance as the action loss function ℓ . This affects the definition of action impurity (Equation 3.6).

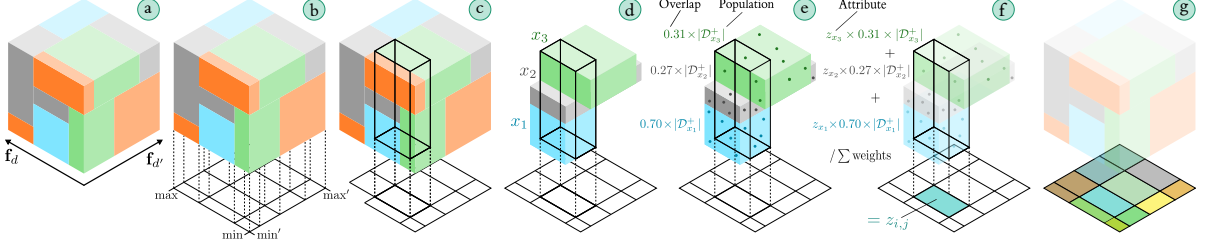


Figure 3.20: Hyperrectangle projection process for $D = 3$. Colours represent the leaf-level summary statistic to be visualised, such as mean action or value, or observation density. Notice that the final colour of each rectangular area is an average of the leaf cuboids above it.

Now imagine simultaneously extruding one of these rectangular areas along each of the $D - 2$ orthogonal feature axes. The volume swept (which we call a *core*) is itself a D -dimensional hyperrectangle which intersects at least one of the hyperrectangular leaves. For each $i \in \{2..|\mathcal{B}^d|\}$, $j \in \{2..|\mathcal{B}^{d'}|\}$, the set of intersected leaves can be identified as

$$(3.33) \quad X_{i,j} = \left\{ x \in \mathcal{X} : (\text{lo}(x, d) \leq \mathcal{B}_i^d) \wedge (\text{hi}(x, d) \geq \mathcal{B}_{i-1}^d) \wedge (\text{lo}(x, d') \leq \mathcal{B}_j^{d'}) \wedge (\text{hi}(x, d') \geq \mathcal{B}_{j-1}^{d'}) \right\}.$$

Steps (c) and (d) in Figure 3.20 show the process of extruding a rectangle from the d - d' plane and identifying intersections. In this case, the core intersects three leaves: x_1 , x_2 and x_3 .

Precisely how we proceed from this point depends on which summary statistic is being visualised. For the sake of brevity, we assume a real-valued statistic, denoted generically by z_x for each $x \in \mathcal{X}$.²⁰ For each rectangle in the d - d' plane, identified by boundary indices i and j as above, we effectively marginalise out the $D - 2$ orthogonal dimensions by taking a weighted mean of the statistic values from the intersected leaves $X_{i,j}$. The weight for each leaf $x \in X_{i,j}$, is jointly determined by the number of observations it contains from the dataset, $|\mathcal{D}_x^+|$, and the degree to which its hyperrectangle overlaps with the core. Concretely, the projected summary statistic for rectangle i, j is defined as

$$(3.34) \quad z_{i,j} = \frac{\sum_{x \in X_{i,j}} w_{i,j,x} \cdot z_x}{\sum_{x \in X_{i,j}} w_{i,j,x}}, \quad \text{where } w_{i,j,x} = |\mathcal{D}_x^+| \cdot \frac{\mathcal{B}_i^d - \mathcal{B}_{i-1}^d}{\text{hi}(x, d) - \text{lo}(x, d)} \cdot \frac{\mathcal{B}_j^{d'} - \mathcal{B}_{j-1}^{d'}}{\text{hi}(x, d') - \text{lo}(x, d')}.$$

Due to the way in which the rectangular tiling of the d - d' plane is defined, the core must contain exactly zero boundaries along either d or d' , so both fractions in the formula for $w_{i,j,x}$ are always ≤ 1 . This part of the process is illustrated by steps (e) and (f) of Figure 3.20, and step (g) shows the result of repeating for all remaining rectangles on the plane, thereby creating the PDP visualisation. The key assumption behind this core-and-average approach to projection is

²⁰This applies to the projection of value and univariate continuous action predictions, as well as each of the three impurities. For derivative vectors or multivariate continuous actions, the averaging can be performed on an elementwise basis. For discrete actions, we cannot take a weighted mean so instead propose to add up the per-action counts for overlapping leaves, weighted by their overlap proportions, and then visualise the modal action for each rectangle. For density visualisations, the population factor $|\mathcal{D}_x^+|$ in Equation 3.34 is replaced by the leaf's 'volume' in $\phi(\mathcal{S})$, as defined in Section 3.9.1. As an implementation detail, we find that it is best to use a logarithmic colour map for density plots, since this statistic can vary over many orders of magnitude.

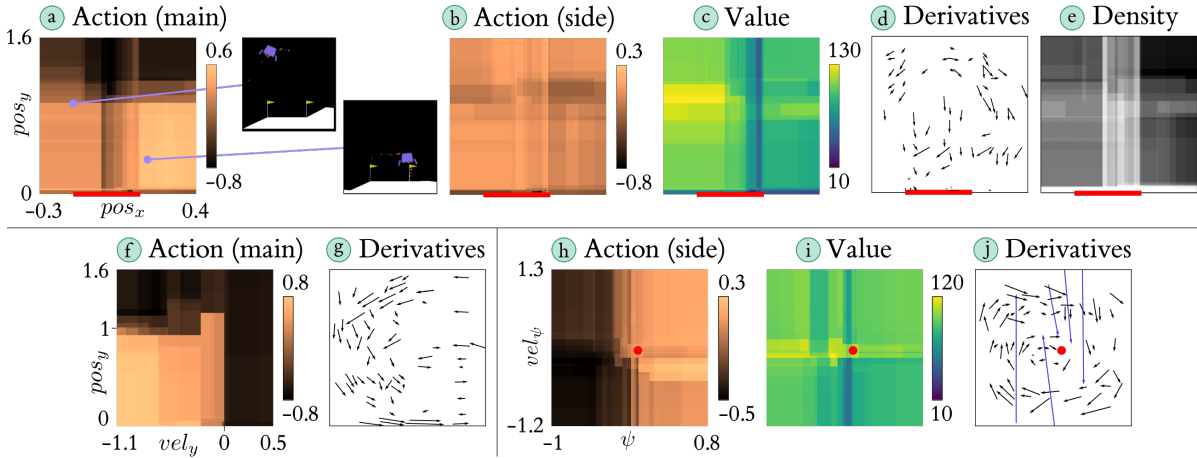


Figure 3.21: Partial dependence plots for the LunarLander policy.

that observations are close to uniformly distributed within leaf hyperrectangles, so that each $w_{i,j,x}$ is an unbiased estimate of the number of observations from x within the core, and each z_x is an unbiased estimate of the summary statistic for those observations.

Figure 3.21 shows PDPs generated from the 450-leaf tree for the LunarLander SAC policy. The upper row of plots are in the pos_x - pos_y plane (landing zone shown in red). From (a), we learn that the agent tends to fire the main engine less at high altitudes, allowing the lander to freefall. This is sensible given the penalty the reward function places on fuel burn. The density plot (e) indicates that the agent spends most time in a central column above the landing zone, and on the ground where the policy makes slow positional corrections. The value and derivatives plots (c) and (d) reveal that despite the environment being symmetric, the agent obtains higher value when landing from the left, and takes a less curved route when doing so. This suggests an asymmetric bias in the policy, which will have emerged at some point during its training process. The side engine plot (b) has weaker trends, but the dark band around $pos_y = 1$ (indicating the engine tends to fire to the left) may partly explain the wider landing approaches on the right.

Moving to the second row of Figure 3.21, a PDP for action in the pos_y - vel_y plane (f) shows hard thresholds in main engine activation at $vel_y \approx 0$ and $pos_y \approx 1$, suggesting that the policy’s use of this engine is close to binary, and heavily correlated with these key thresholds in vertical position and speed. The derivative plot in the same plane (g) indicates a C-shaped vertical velocity profile, consistent with a soft and controlled landing after freefall. In the ψ - vel_ψ plane, we see that the side engine (h) fires in an intuitive way to maintain stability: to the left when the lander is rotated anticlockwise, and vice versa (note that the red dot indicates the origin). In the same plane, the PDP of value (i) is highest when $vel_\psi \approx 0$, showing that slow changes in angle tend to preempt better performance, and the derivative plot (j) shows that the lander has pendulum-like rotational dynamics, aside from several leaves (purple) where vel_ψ jumps. These jumps are likely caused by rapid changes in side engine activation.

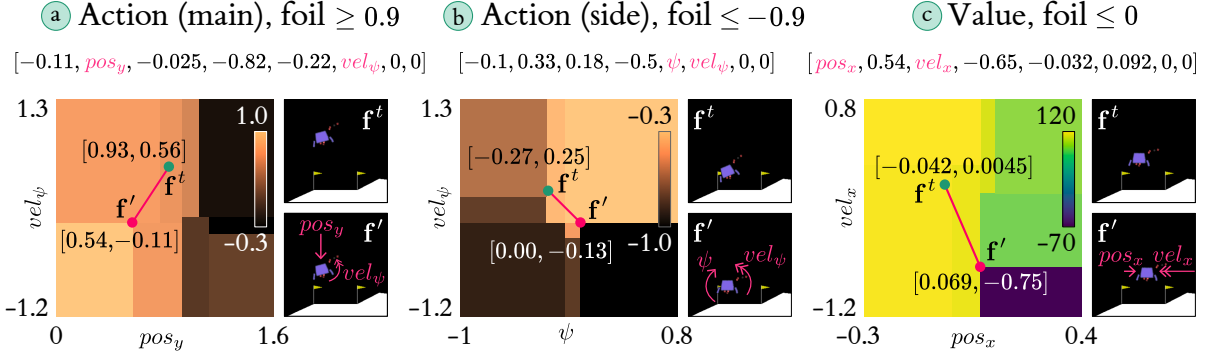


Figure 3.22: Displaying visual counterfactuals for the LunarLander policy on ICE plots.

3.10.2 Hyperrectangle Slicing and Visual Counterfactuals

We now discuss the slicing visualisation method, which is complementary to projection. Here, we again choose two features to visualise over, d and d' . For each remaining feature $d'' \in \{1, \dots, D\} \setminus \{d, d'\}$, we specify a single threshold, denoted by $\text{slice}_{d''}$. The set of thresholds defines the offset of an axis-aligned hyperplanar cross-section through $\phi(\mathcal{S})$, which intersects a subset of the leaves. Here we do not have to handle overlaps, and can display the rectangular cross-sections of the intersected leaves directly. This creates an individual conditional expectation (ICE) plot, which is also well established in the interpretability literature [180].

The slicing process is more straightforward than projection. We simply need to identify the subset of leaves intersected by the slicing hyperplane:

$$(3.35) \quad X_{\text{slice}} = \bigcap_{d'' \in \{1, \dots, D\} \setminus \{d, d'\}} \left\{ x \in \mathcal{X} : (\text{lower}(x, d'') \leq \text{slice}_{d''}) \wedge (\text{upper}(x, d'') \geq \text{slice}_{d''}) \right\}.$$

Each $x \in X_{\text{slice}}$ is then visualised as a rectangle with boundaries at $\text{lower}(x, d)$, $\text{upper}(x, d)$, $\text{lower}(x, d')$ and $\text{upper}(x, d')$, and coloured according to a chosen summary statistic.²¹

One use of ICE plots is to help illustrate counterfactual explanations for which the initial feature vector f^t and minimal counterfactual f' differ in ≤ 2 features. Here, a slicing hyperplane can be defined by the values of the unchanged features. Examples for the LunarLander policy are shown in Figure 3.22. In each case, the foil is specified in the subplot heading, and the slicing hyperplane below that. These plots not only display the minimal state change required to realise the foil condition, but reveal some of the surrounding feature space. This gives an indication of the counterfactual’s robustness to perturbations, as well as other (non-minimal) counterfactuals that would also realise the foil condition. The fact that such visualisations are

²¹Although we have presented projection and slicing as two distinct visualisation methods, in reality it is possible to smoothly transition between the two. Starting from the projection method as described, we achieve this by permitting the core extrusion along feature d'' to be limited between two thresholds $\text{min}_{d''}$ and $\text{max}_{d''}$. This allows us to visualise projections from only those leaves within a hyperrectangular subset of $\phi(\mathcal{S})$ rather than the entire space. With this formulation, it is easy to see that slicing results from the limiting case where $\text{min}_{d''} = \text{max}_{d''} = \text{slice}_{d''}$, for all $d'' \in \{1, \dots, D\} \setminus \{d, d'\}$.

only possible when counterfactuals use ≤ 2 features is another justification for prioritising the sparsity-oriented 0-norm over the 2-norm in our definition of minimality.

To briefly discuss each case in Figure 3.22, (a) considers a state where the lander is mid-height and left of centre (see rendered image to the right). The foil asks why the main engine is not strongly activated (≥ 0.9). We find that the engine would be activated in this way if the lander were lower to the ground ($pos_y \leq 0.54$) and rotating slightly anticlockwise ($vel_\psi \leq -0.11$). In the scenario studied in (b), we find that the side engine would be fired strongly to the left (≤ -0.9) if the lander were rotating anticlockwise from a level angle. Finally, the state in (c) has high predicted value, but the counterfactual suggests that value would be far lower (i.e. the lander would be expected to land less successfully) if it were moving right-to-left from the other side of the centreline ($pos_x = 0$). This local example is another signal of the policy’s asymmetry.

3.10.3 Hypothetical Trajectories

Hypothetical trajectories can also be generated in this environment. Although the derivative alignment optimisation can be done in the full 8D feature space, the trajectories can only be visualised over two features at a time. Figure 3.23 contains some examples. Rather than showing a single trajectory between two leaves, we display all possible trajectories between leaves contained within a start zone $\subset \phi(\mathcal{S})$ (blue/orange) and a goal zone $\subset \phi(\mathcal{S})$ (red), demonstrating the distribution of possible paths taken by the agent. Practically, this set of trajectories is obtained by performing a multi-source, multi-destination Dijkstra search between the two sets of leaves $X_0 = \{x \in \mathcal{X} : (x \cap \text{start zone}) = x\}$ and $X_{\text{goal}} = \{x \in \mathcal{X} : (x \cap \text{goal zone}) = x\}$. Popular Dijkstra implementations are capable of doing this efficiently (we use the `networkx` Python library). This provides a powerful generalisation of the single start/goal leaf case presented earlier, enabling us to query hypothetical trajectories between manually specified start and goal conditions that do not directly correspond to the boundaries of individual leaves.²²

In Figure 3.23, The opacity of each trajectory is proportional to its likelihood according to the leaf transition probabilities. (a) clarifies the prior observation that approaches from the right side (orange, c.f. blue) are wider and more curved, and shows that they occasionally miss the landing zone (thick red line) altogether. Thereafter, the lander must activate its side engine to ‘shuffle’ along the ground into position; a major source of lost value. Similarly, (b) confirms that the lander’s vertical velocity profile tends to be C-shaped, reaching its maximum around $pos_y = 0.9$ before decreasing approximately linearly. The final plot (c) provides the most novel insight. If rotated anticlockwise ($\psi \leq -0.5$), the lander’s return to a stable, neutral orientation is direct and overdamped. From clockwise ($\psi \geq 0.5$), trajectories back to neutrality tend to

²²There is a connection between our general formulations of hypothetical trajectories and counterfactual explanation. In the former, we are given two sets of leaves X_0 and X_{goal} and search for the ‘best’ (i.e. most dynamically realistic) multi-segment linear path between them. In the latter, we are given a single point and a set of leaves X_{foil} and search for the ‘best’ (i.e. minimal) single-segment linear path between them. This hints at the potential for a unifying formalism.

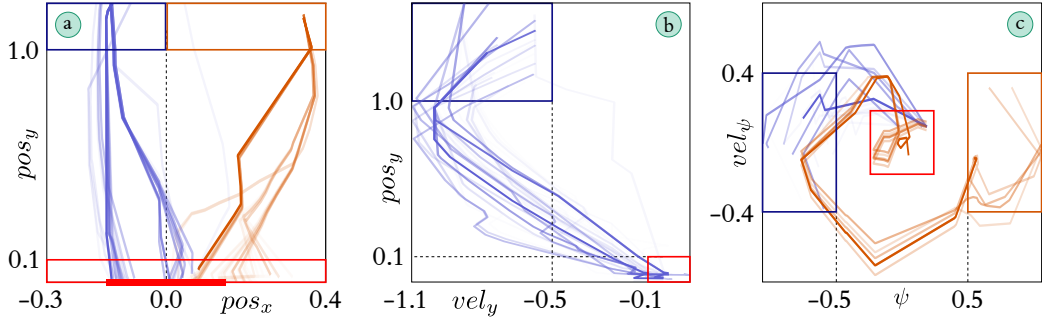


Figure 3.23: Hypothetical trajectories for the LunarLander policy.

overshoot; a classic indicator of a poorly-tuned controller. This is further evidence that despite attaining higher expected reward than all others in the Baselines Zoo repository, the SAC policy exhibits chronic asymmetries that may cause problems in deployment.

Finally, we note that the ASQ-IT framework [9], which was published after the work in this chapter, is quite similar to our method of querying agent trajectories based on start and goal conditions. In that work, the conditions are specified in linear temporal logic, and can be complemented by constraints on how the agent is allowed to move between the start and goal. The key point of difference is that ASQ-IT searches through a dataset of real agent trajectories rather than synthesising novel ones using a learnt transition model.

3.11 Conclusion

In this chapter, we have used the language of tree abstractions to build interpretable models of agents based on observations of their behaviour. We initially considered policy-only models, which have precedence in the existing literature. We contributed a novel method for programmatically generating interpretable features, explored tree-based techniques for producing textual explanations, and adapted these techniques to explain changes in agent actions over an extended trajectory. We also examined the relationships between different measures of the performance or accuracy of a policy-only model across different evaluation environments (track topologies). We then proposed TRIPLETREE, a multiattribute tree abstraction that combines policy information with summaries of an agent’s value function and state dynamics, providing a more holistic view of its behaviour. In exploring the potential of the TRIPLETREE model across two evaluation environments, we developed novel methods for visualisation and hypothetical trajectory generation based on user-specified queries. The models and experiments in this chapter have several limitations, which create opportunities for further work:

- The operator-based feature generation method was conceived and evaluated in the traffic simulator environment, but we hypothesise that it is applicable in a wide range of domains. It would be valuable to try implementing it in other contexts. As an end goal, we can envisage a flexible interactive interface for allowing users to specify interpretable base

features, operators, and constraints on their combination, and visualise how the inclusion or exclusion of different features affects the growth of a tree model.

- Our policy trees were grown and pruned for the task of accurate action prediction via the CART algorithm, but then evaluated on a range of other metrics such as on-policy performance and MTBF, which we showed to be imperfectly correlated. Further work could develop more sophisticated split criteria that directly optimise these other metrics.
- TRIPLETREE combines its various leaf impurity measures by simple summation, weighted by a constant vector $\theta \in \mathbb{R}_+^3$. Although we found this to be satisfactory, other options could be worth exploring, such as allowing the weighting to vary across different regions of the state feature space according to a user’s interest.
- As part of our examination of temporal explanations, we developed a general criterion for minimal, non-misleading explanations based on the MBB constraint, but only devised a practical algorithm for 2D feature spaces. An efficient generalisation to higher dimensionalities would be valuable.
- Our implementation of TRIPLETREE collects statistics on an agent’s transition dynamics between leaves of the tree, which enables hypothetical trajectory generation, but this functionality is somewhat peripheral and the tree is not grown or pruned with these statistics in mind. In the next chapter, we develop a much more principled tree-based model for summarising an agent’s transition dynamics.
- Both models in this chapter provide insight into an agent’s behaviour under a single, fixed policy, but do not capture how that behaviour changes as the agent learns over time. This is also a major focus of the next chapter.

Chapter 4

Tree Models of Agent Learning

Based on: “Summarising and Comparing Agent Dynamics with Contrastive Spatiotemporal Abstraction”, presented at 2022 IJCAI Workshop on Explainable Artificial Intelligence.

4.1 Introduction

In the preceding chapter, we developed tree abstractions that summarise and explain a single agent’s fixed distribution of behaviour based on observational data. Depending on the application and intended audience of the model, such insight may be sufficient. However, a fundamental characteristic of many contemporary artificial agents is their ability to learn from experience, altering their action-selection policy, and thus the behaviour distribution we observe. When such changes occur, any existing model of the agent becomes an obsolete and misleading representation of the current or future behaviour. Moreover, stakeholders seeking insight into the mechanisms of learning, particularly researchers, engineers, and auditors, may be interested in comprehending the nature, causes, timings, and co-occurrence of *the changes themselves*.

Evidence of an agent’s learning trends may be more informative than just its current policy, particularly from the standpoint of safety validation and iterative design. For example, knowing that unsafe or inefficient behaviours have been explored and rejected provides some confidence that the agent will avoid making the same mistakes in future. Additionally, there is evidence that RL agents in particular exhibit “*Aha!*” moments, similar to those experienced by human learners, where internal representations rapidly transform to improve task performance [41]. Understanding how and when such moments arise would be a critical component of a functional theory of how agents learn in practice across environments and tasks, that extends beyond the idealised cases covered by formal proofs. We have already seen evidence of the potential value of understanding an agent’s learning process in Section 3.10, where our analysis revealed an asymmetry in a baseline LunarLander policy. If we had information on when and how this asymmetry emerged as the soft actor-critic RL agent was learning, we might be able to find ways to prevent or correct it during subsequent training runs.

In a 2021 position paper [108], Hammer et al. argue that the problem of summarising and explaining the incremental changes during a machine learning process is heavily underexplored. They identify several sub-problems, such as (1) finding interpretable and flexible representations of change, (2) quantifying the magnitude of changes in a computationally efficient manner, and (3) determining when and how frequently changes should be recorded, which is effectively an issue of change point detection [5]. In this chapter, we address all three sub-problems:

1. For our interpretable representation, we retain our thesis-wide focus on tree abstractions, learnt from observational data. As ever, there is a question of which summary statistics to compute and store in the tree. This affects the kinds of change that can be represented. While it would be reasonable to develop abstract models of the temporal changes in an agent’s value function or action-selection policy, we reflect that some of the most interesting and novel parts of the preceding chapter involved analysing an agent’s state dynamics. For this reason, the core of the model presented in this chapter is an abstract representation of agent dynamics at each point in time as a set of transition probabilities between the leaves of the tree (here referred to as *abstract states*). Since no action or reward information is required to build this model, it is by some distance the most context-agnostic one in this thesis, with potential applications outside of agent interpretability.
2. To quantify how the agent changes, we adopt an information-theoretic divergence measure between the probabilities of transitioning from one abstract state to another at different points in time. In turn, we use this measure as an objective to maximise when constructing the abstraction itself. The effect of using this objective is that the abstract transition model selectively highlights significant changes in behaviour in a highly compressed form.¹
3. The same divergence measure can be used to address the sub-problem of change point detection. Instead of modelling agent dynamics separately at every time point (which, among other issues, would likely overwhelm a human observer), we partition the agent’s learning history into a reduced number of contiguous time windows in such a way that the inter-window transition divergence is maximised. The practical algorithm for this *temporal abstraction* stage is structurally very similar to that used for state abstraction.

As the resultant abstractions are explicitly optimised to highlight differences, they enable us to take a contrastive perspective on explaining the agent’s learning process, by identifying large or salient differences in the transition probabilities of two or more time windows. Such contrastive explanations are favoured in both the psychological [115, 160, 254] and computational [176] literature, where they are argued to be natural and human-like.

¹There is an interesting duality between this approach and a prior effort to encourage agents to exhibit diverse behaviour by maximising the mutual information between a ‘skill indicator’ and distributions over state visitation [83]. In that work, new policies are learnt to maximise dynamical diversity in a given state space. In contrast, we learn a new (abstract) state space that maximises the measured diversity of given policies.

After introducing our proposed theory of contrastive spatiotemporal abstraction in generic terms, we describe a practical tree-based algorithm for real vector state spaces. We deploy it to summarise the nonstationary dynamics of RL agents in two continuous control environments (maze navigation and lunar landing) with the aid of tree diagrams, transition graph visualisations, prototype trajectories and inter-window counterfactuals. We then introduce a series of modifications to the tree growth and pruning stages of our learning algorithm and explore their implications for performance and computational efficiency.

4.1.1 Related Work

Existing work on summarising agent change is limited. The few prior examples include displaying nonstationary behavioural data directly via a visual analytics dashboard [262] and using representative state-action pairs to summarise a policy at various key moments during learning [59]. The DISAGREEMENTS algorithm [8] and its extensions [76, 92] take an explicitly contrastive perspective, synthesising exemplar trajectories over which different policies’ actions differ, but only works in the case of two policies, and provides only local examples rather than a global model of disagreement or change. In [187], an active querying scheme is proposed for finding and representing temporal changes in classical planning algorithms. While abstract transition models have been used to summarise agent behaviour before [60, 171, 249, 272], we believe that our work is the first to handle multiple or nonstationary policies, and to develop principled theories and algorithms based on the notion of contrastive explanation.

4.2 Theory of Contrastive Spatiotemporal Abstraction

Consider a learning agent whose action-selection policy evolves over a sequence of k update steps $\Pi = (\pi_1, \dots, \pi_k)$, where $\pi_i : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, $\forall i \in \{1, \dots, k\}$. When deployed in the environment, each of these policies induces a particular distribution of behaviour. This can be formalised as a Markov chain over the state space with conditional transition probabilities $P_i(s'|s) = \sum_{a \in \mathcal{A}} \pi_i(a|s)T(s'|s, a)$, $\forall s \in \mathcal{S}$, $\forall s' \in \mathcal{S}$, where $T : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the environment’s (unchanging) dynamics function. The policy updates therefore implicitly generate a sequence of such Markov chains $\mathcal{P} = (P_1, \dots, P_k)$. Suppose we do not have direct access to \mathcal{P} , but can observe the history of interactions performed by the agent during its learning. We thereby assemble a transition dataset \mathcal{D} comprising elements of the form (i, s, s') , where i is the index of the current policy (which is incremented by 1 every time an update is made) and s, s' is a pair of successive states sampled from its induced Markov chain.²

²Technically, our theoretical framework assumes that observed transitions are independent, which is not true for sequential observations of a Markovian system. This issue is mitigated by observing many independently-initialised episodes in an episodic environment, or by running each policy until it attains its stationary distribution in a non-episodic environment. In the episodic environments used in our experiments, we constrain the learning agents to only perform updates between, rather than within, episodes, thereby guaranteeing that the observed transitions from each policy are independent of the dynamics of all previous ones.

Notice that unlike the methods presented in the previous chapter, assembling the transition dataset does not require access to the agent’s actions, or the outputs of the environment’s reward function, if one exists. This means the abstraction theory and algorithms that follow can in principle be applied to any sequence of Markovian dynamical systems with a common state space. We discuss the implications of this extreme context-agnosticism in Section 4.8.

An example of a small transition dataset \mathcal{D} for $k = 3$ policies is shown in Figure 4.1 (a). Our goal in this chapter is to use \mathcal{D} to model the major dynamical changes that occur throughout the agent’s learning process. To do so, we propose to give the transition data a more compact, interpretable representation through the use of abstraction. Following the notation used in previous chapters, let an abstraction of the state space be a set $\mathcal{X} = \{x_1, \dots, x_m\}$, whose m elements (here called *abstract states*) partition \mathcal{S} , i.e. $\bigcup_{x \in \mathcal{X}} x = \mathcal{S}$ and $\bigcap_{x \in \mathcal{X}} x = \emptyset$. Figure 4.1 (b) shows a possible $m = 3$ -state abstraction with the transition data overlaid. Note that we have drawn abstract states with arbitrary geometries because the theory in this section does not rely on the axis-aligned partition constraint used in our practical tree methods.

Given such an abstraction, we can summarise the transition data \mathcal{D} via a $k \times m \times m$ array of abstract transition counts $N^{\mathcal{X}}$, as shown in Figure 4.1 (c). For each policy $i \in \{1, \dots, k\}$ and pair of states $x \in \mathcal{X}, x' \in \mathcal{X}$, the transition count from x to x' is defined as

$$(4.1) \quad N_{i,x,x'}^{\mathcal{X}} = |\{(j, s, s') \in \mathcal{D} : (j = i) \wedge (s \in x) \wedge (s' \in x')\}|.$$

Let $N_i^{\mathcal{X}}$ be the $k \times k$ matrix of transition counts for policy i . Normalising $N_i^{\mathcal{X}}$ by its grand sum yields an empirical joint distribution over abstract transitions $J_i^{\mathcal{X}}$, and separately normalising each row gives a conditional distribution $P_i^{\mathcal{X}}$. $J_2^{\mathcal{X}}$ and $P_2^{\mathcal{X}}$ are shown in Figure 4.1 (d). The latter is of particular interest as it defines a Markov chain over the abstract states \mathcal{X} . Abstract Markov chains can be visualised as transition graphs, as shown in Figure 4.1 (e). Such visualisations have been used in several prior works to provide simplified summaries of agent dynamics in high-dimensional environments [171, 249]. We hypothesise that provided we always remain aware of the epistemic relationship between $P_i^{\mathcal{X}}$ and P_i (i.e. the former is a compressed model of the latter, estimated from limited data) and if \mathcal{X} is carefully constructed, comparative analysis of the abstract Markov chains can provide meaningful and accurate insight into the agent’s changing dynamics. For instance, we can apply established methods to analyse properties such as convergence, stability and cycles in the abstract state space, which may not be possible or tractable in the original state space \mathcal{S} .

What do we mean by the term “*carefully constructed*” in the preceding paragraph? Clearly, there is a tradeoff around the number of abstract states m , with the extrema $\mathcal{X} = \{\mathcal{S}\}$ ($\therefore m = 1$) and $\mathcal{X} = \{s : s \in \mathcal{S}\}$ ($\therefore m = |\mathcal{S}|$) respectively providing a degenerate model or zero interpretability gain. But beyond this, we can reasonably expect some m -sized state abstractions to be more effective than others at capturing salient features of the underlying dynamics. We investigate this hypothesis by postulating an objective for constructing \mathcal{X} .

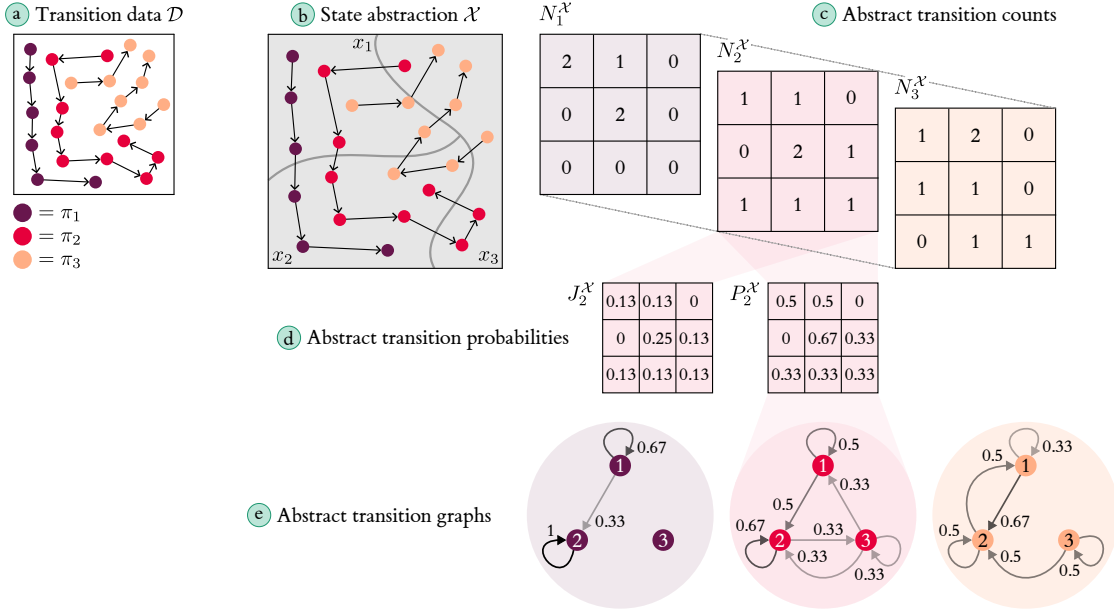


Figure 4.1: Application of state abstraction to a $k = 3$ -policy transition dataset.

4.2.1 The Contrastive Objective

According to several influential theories [115, 160, 254], human-like explanations of observed phenomena are constructed by searching for simple points of contrast with alternative cases. This in turn implies that a common representation of multiple phenomena has explanatory value to the extent that it preserves information about their mutual differences, while ignoring irrelevant details. This selective emphasis on contrasts has been adopted as a recommendation for machine learning interpretability work [177] and underlies the previously-cited DISAGREEMENTS algorithm for pairwise policy comparison [8].

Motivated by this precedent, we propose a new definition of abstraction efficiency (recall Section 2.4 as the origin of this term). In the present context, we say that an abstraction \mathcal{X} is efficient to the extent that it maximises our ability to discriminate between policies $i \in \{1, \dots, k\}$ based on the abstract state transition probabilities they generate, while being as compact as possible (small m). Such abstractions would be optimised for queries of the form “which policy is most likely to have produced this abstract transition $x \rightarrow x'$?” To perform such discrimination effectively, \mathcal{X} must consist of abstract states that are visited and transitioned between with different probabilities by the different policies.

To formalise this definition, let us consider the sequence of abstract transition models, defined by the joint probabilities $J_1^{\mathcal{X}}, \dots, J_k^{\mathcal{X}}$, as generative distributions from which transitions can be sampled.³ Applying Bayes’ rule, the posterior probability of a policy i given a transition

³It is important to develop our theory based on joint, rather than conditional, transition probabilities because it ensures that the divergence measures that follow are always well-defined. If a policy i never visits a particular abstract state x , $P_{i,x,x'}$ cannot be computed for any x' , but $J_{i,x,x'}$ can be unproblematically defined as zero.

x, x' sampled from its abstract model is

$$(4.2) \quad \Pr(i|x, x') = \frac{\rho_i J_{i,x,x'}^{\mathcal{X}}}{\sum_{j=1}^k \rho_j J_{j,x,x'}^{\mathcal{X}}}.$$

The vector $\boldsymbol{\rho} \in [0, 1]^k$, $\sum_{i=1}^k \rho_i = 1$ is a prior weighting over policies, which may be manually specified or derived from data (e.g. in proportion to the number of observations in \mathcal{D}). Taking the prior-weighted expectation of the log posterior over all policies and transitions, we obtain:

$$(4.3) \quad \begin{aligned} \mathbb{E}_{i,x,x'} [\log \Pr(i|x, x')] &= \sum_{i=1}^k \rho_i \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} J_{i,x,x'}^{\mathcal{X}} \left[\log \frac{\rho_i J_{i,x,x'}^{\mathcal{X}}}{\sum_{j=1}^k \rho_j J_{j,x,x'}^{\mathcal{X}}} \right] \\ &= \sum_{i=1}^k \rho_i \log \rho_i \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} J_{i,x,x'}^{\mathcal{X}} \stackrel{=1}{=} 1 + \sum_{i=1}^k \rho_i \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} J_{i,x,x'}^{\mathcal{X}} \log J_{i,x,x'}^{\mathcal{X}} \\ &\quad - \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} \left(\sum_{i=1}^k \rho_i J_{i,x,x'}^{\mathcal{X}} \right) \log \left(\sum_{j=1}^k \rho_j J_{j,x,x'}^{\mathcal{X}} \right) \\ &= -\mathcal{H}(\boldsymbol{\rho}) - \sum_{i=1}^k \rho_i \mathcal{H}(J_i^{\mathcal{X}}) + \mathcal{H} \left(\sum_{i=1}^k \rho_i J_i^{\mathcal{X}} \right) = \text{JSD}(J^{\mathcal{X}}|\boldsymbol{\rho}) - \mathcal{H}(\boldsymbol{\rho}). \end{aligned}$$

\mathcal{H} is entropy, and JSD denotes the multi-distribution generalisation of the Jensen-Shannon divergence [158], which is equivalent to the mutual information between the prior-weighted mixture distribution $\sum_{i=1}^k \rho_i J_i^{\mathcal{X}}$ and the policy index i . Intuitively, Equation 4.3 tells us that policy discrimination is facilitated by an abstraction that yields abstract transition probabilities $J_1^{\mathcal{X}}, \dots, J_k^{\mathcal{X}}$ which are as different as possible, and also by minimising $\mathcal{H}(\boldsymbol{\rho})$, although this second term is not relevant here since we take $\boldsymbol{\rho}$ to be fixed. As we will verify in our experiments, discrimination is practically facilitated by placing the boundaries of abstract states at points of divergence between the policies, thus disentangling their transitions.

We therefore propose the following *contrastive abstraction* objective. Given a set of state abstractions \mathbb{X} constructible by some well-defined algorithm, the objective is

$$(4.4) \quad \operatorname{argmax}_{\mathcal{X} \in \mathbb{X}} \left[\mathbb{E}_{i,x,x'} \log \Pr(i|x, x') - \alpha(m-1) \right] = \operatorname{argmax}_{\mathcal{X} \in \mathbb{X}} \left[\text{JSD}(J^{\mathcal{X}}|\boldsymbol{\rho}) - \alpha(m-1) \right].$$

Here, $\mathcal{H}(\boldsymbol{\rho})$ is ignored because it is independent of \mathcal{X} , and the second term (weighted by $\alpha > 0$) is a regulariser to incentivise compact abstractions. This term ensures the objective is not maximised by the trivial solution $\mathcal{X} = \{s : s \in \mathcal{S}\}$. It uses $m-1$ rather than m , because this sets the value of the objective for the degenerate one-state abstraction $\mathcal{S} = \{\mathcal{S}\}$ to zero. In expectation, the JSD term increases monotonically but sublinearly with m (see Section 4.4 for a theoretical analysis and empirical demonstration), and $\alpha(m-1)$ is evidently linear, so we should expect their difference to be maximised at an intermediate abstraction size determined by α (larger α favours smaller m).

Figure 4.2 shows two options (A and B) for an $m=2$ -state abstraction given transition data from $k=2$ policies. Option A is a superior state abstraction according to Equation 4.4. With a

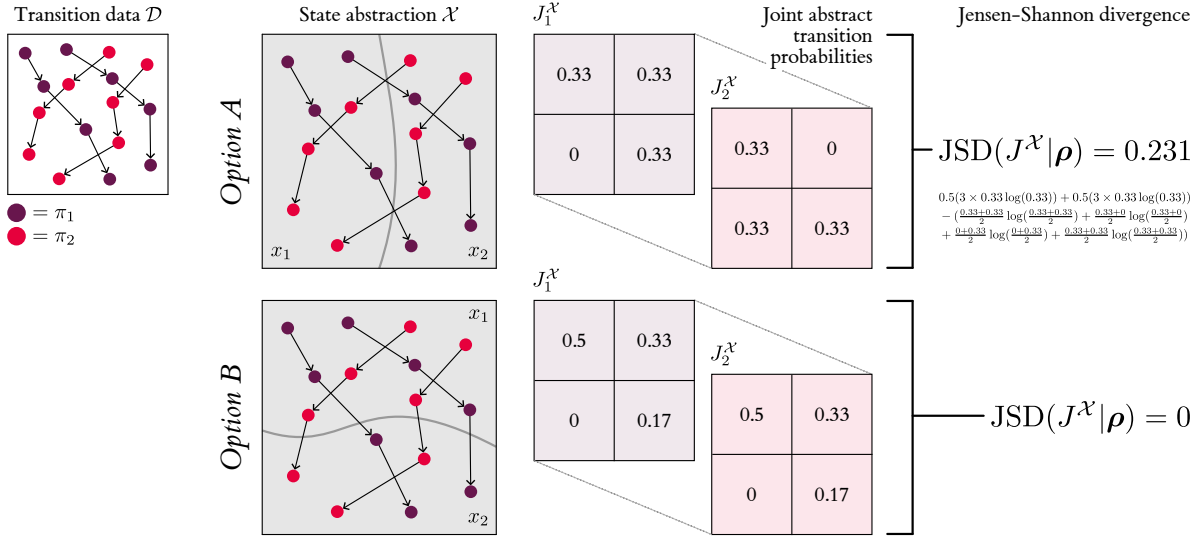


Figure 4.2: Two state abstraction options and their resultant JSD values.

uniform prior $\rho = [0.5, 0.5]$, it gives a JSD of 0.231, compared with exactly zero for option B, while incurring the same regularisation penalty because $m = 2$ is constant. Intuitively, option A preserves the critical difference between the policies: π_1 crosses the abstract state boundary from left-to-right ($x_1 \rightarrow x_2$) while π_2 crosses from right-to-left ($x_2 \rightarrow x_1$). In contrast, option B fails to preserve a difference: the top-to-bottom dynamics of the two policies are identical.

4.2.2 Temporal Abstraction

The preceding discussion concerns a sequence of k policies $\Pi = (\pi_1, \dots, \pi_k)$ and their induced Markov chains $\mathcal{P} = (P_1, \dots, P_k)$, whose dynamics are to be summarised via contrastive abstraction. Recall that for our purposes, the generative origin of this policy sequence is a contiguous process of agent learning. Since modern learning algorithms typically run for many steps and update their policies with high frequency, a likely consequence is that the number of individual policies k is large, and the transition dataset \mathcal{D} contains only a small number of observations of each P_i . Therefore, regardless of our choice of state abstraction \mathcal{X} , the result will be an array of joint probabilities $J^{\mathcal{X}}$ that is not only large (hampering interpretability) but also sparse, meaning the estimated abstract Markov chains will have such high variance as to be practically useless as summary models of \mathcal{P} .

However, for many practical learning agents, policy changes are incremental and gradual, so it is reasonable to assume approximate stationarity over short timescales. We therefore propose to reduce both the size and the sparsity of $J^{\mathcal{X}}$ by applying a layer of temporal abstraction, which approximates the dynamics as piecewise constant within $n \ll k$ disjoint and exhaustive time windows $\mathcal{W} = \{\tau_1, \dots, \tau_n\}$. Each τ_w is parameterised by lower and upper bounds l_w, u_w , and the following constraints hold: $l_1 = 1$; $u_n = k + 1$; $l_w < u_w$, $\forall w \in \{1, \dots, n\}$; $u_w = l_{w+1}$, $\forall w \in \{1, \dots, n - 1\}$. These can be used to aggregate $J^{\mathcal{X}}$ along its first axis to form

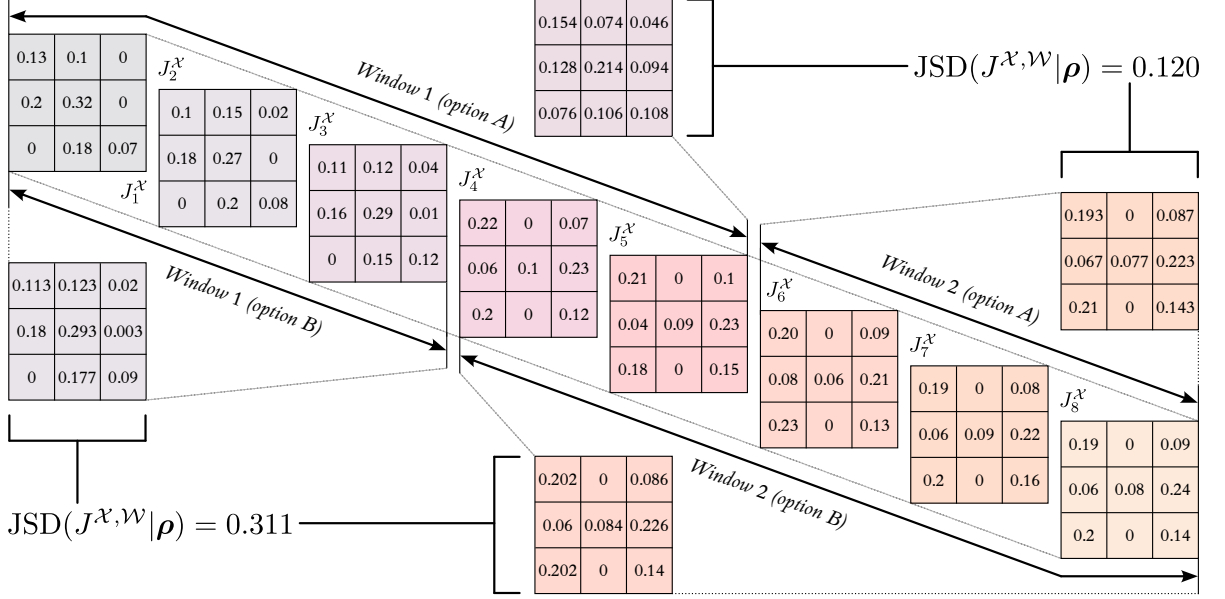


Figure 4.3: Two temporal abstraction options and their resultant JSD values.

a smaller and less sparse $n \times m \times m$ array $J^{\mathcal{X}, \mathcal{W}}$, where $\forall w \in \{1, \dots, n\}, x \in \mathcal{X}, x' \in \mathcal{X}$,

$$(4.5) \quad J_{w,x,x'}^{\mathcal{X}, \mathcal{W}} = \frac{1}{\rho_w^{\mathcal{W}}} \sum_{i=l_w}^{u_w-1} \rho_i J_{i,x,x'}^{\mathcal{X}},$$

and $\rho_w^{\mathcal{W}} = \sum_{i=l_w}^{u_w-1} \rho_i$ aggregates the elements of the prior vector for each window. By varying both n and the window boundaries in \mathcal{W} , we obtain different sequences of spatiotemporally abstracted transition probabilities, even if \mathcal{X} remains constant. For example, Figure 4.3 shows two options (A and B) for aggregating abstract transition probabilities from a $k = 8$ -policy sequence into $n = 2$ windows, and the results of applying Equation 4.5 (with uniform ρ).

As with \mathcal{X} , a question arises as to why one set of windows \mathcal{W} should be chosen over another. Intuitively, a temporal abstraction is informative for contrastive analysis if it contains windows of behaviour which are internally consistent, but between which there are significant differences. We suggest that the desired property can be achieved using an *identical objective* to that of the previous section: maximising the Jensen-Shannon divergence between joint transition distributions, but now of the windows rather than the individual policies. As always, we wish to incentivise compact abstractions (small n), so add another regularisation term to the objective:

$$(4.6) \quad \operatorname{argmax}_{\mathcal{X} \in \mathbb{X}, \mathcal{W} \in \mathbb{W}} \left[\operatorname{JSD}(J^{\mathcal{X}, \mathcal{W}} | \rho^{\mathcal{W}}) - \alpha(m-1) - \beta(n-1) \right],$$

where \mathbb{W} is the set of all possible window-based abstractions, and $\beta > 0$ is a hyperparameter. Theoretical and empirical results again indicate that JSD is sublinear in n (see Section 4.4). As with m , this means that the objective will be maximised at a finite value of n determined by β .

In the example shown in Figure 4.3, option B is a superior temporal abstraction according to Equation 4.6. It gives an inter-window JSD of 0.311, compared with 0.120 for option A, while having the same regularisation penalty because $m = 3$ and $n = 2$ are constant.

4.3 Tree Abstraction Algorithms

We now present practical algorithms for contrastive abstraction of transition data \mathcal{D} produced by a learning agent. In doing so, we specialise from the general framing of the preceding section to the specific context studied in this thesis, by assuming that the state space can be represented by vectors of interpretable real-valued features $\phi(\mathcal{S}) = \mathbb{R}^D$. We can therefore adopt our favoured approach of axis-aligned tree abstraction, opening the door to efficient learning algorithms, and yielding hyperrectangular abstract states (i.e. leaves of the tree) whose simple geometry aids interpretability.⁴

As in the CART-based algorithms used in the previous chapter, the central idea of our approach is to recursively partition $\phi(\mathcal{S})$ with axis-aligned hyperplanes according to a greedy selection criterion. The top-down partitioning approach imposes a structural bias towards compact abstractions and means the algorithms need only consider solutions up to some acceptable maximum size. The core algorithm, designed to optimise the objective in Equation 4.4 given an arbitrary sequence of k policies, is referred to as contrastive state abstraction (CSA). The extension for aggregating incremental policy learning into a reduced number of time windows is contrastive spatiotemporal abstraction CSTA, which first calls CSA before running an algorithmically-similar temporal partitioning loop to find windows that maximise Equation 4.6. In this section, we provide a high-level overview of the algorithms, followed by annotated pseudocode and further technical details.

The two-stage strategy described below was used to produce the main results in Sections 4.5 and 4.6. In more recent work presented in Section 4.7, we compare this approach to a large number of alternatives in terms of their performance on the objective in Equation 4.6. We find that several of the more sophisticated strategies outperform this one across a range of datasets. We describe only the two-stage approach here for the sake of simplicity and faithfulness to our main results, and defer discussion of the possible improvements to Section 4.7.

4.3.1 Contrastive State Abstraction (CSA)

Suppose that we have an existing tree-structured state abstraction \mathcal{X} , consisting of m hyperrectangles in the state feature space $\phi(\mathcal{S}) = \mathbb{R}^D$, and have precomputed the joint abstract transition probabilities $J^{\mathcal{X}}$ by Equation 4.1 and normalisation.

Now consider splitting an abstract state $x \in \mathcal{X}$ into two by placing a hyperplane at a threshold $c \in \mathbb{R}$ along feature $d \in \{1, \dots, D\}$, exactly as in CART. All transitions in \mathcal{D} into, out of and within the abstract state will be redistributed between the two new abstract states, $x^{(d \geq c)}$ and $x^{(d < c)}$, in a way that conserves their overall sum. For example, for an

⁴As a technical note, for any $\phi : \mathcal{S} \rightarrow \mathbb{R}^D$, a partition of $\phi(\mathcal{S})$ is isomorphic to a partition of \mathcal{S} . This is because any set $x \subseteq \mathcal{S}$ maps to a unique set $x_\phi \subseteq \phi(\mathcal{S}) = \{\phi(s) : s \in x\}$, and vice versa, i.e. $x = \{s : \phi(s) \in x_\phi\}$. In a slight abuse of notation permitted by this isomorphism, we switch to using the term ‘‘abstract state’’ and the symbol ‘‘ x ’’ to refer to subsets of $\phi(\mathcal{S})$ from this point onwards.

observation $(i, s, s') \in \mathcal{D}$ where both $\phi(s) \in x$ and $\phi(s') \in x$, the new transition may be either $x^{(d \geq c)} \rightarrow x^{(d \geq c)}$, $x^{(d \geq c)} \rightarrow x^{(d < c)}$, $x^{(d < c)} \rightarrow x^{(d \geq c)}$ or $x^{(d < c)} \rightarrow x^{(d < c)}$ depending on where the two feature vectors $\phi(s)$ and $\phi(s')$ fall relative to the partitioning hyperplane. These updated transitions can be represented by an enlarged array of joint transition probabilities with shape $k \times (m + 1) \times (m + 1)$, denoted by $J^{\mathcal{X} \rightarrow (x, d, c)}$, which can be computed efficiently since (in the general case) most of its values are unchanged from $J^{\mathcal{X}}$.

This operation of splitting an abstract state in two increases the efficiency of the abstraction according to the objective in Equation 4.4 by an amount equal to the increase in JSD (which is guaranteed to be non-negative), minus the regularisation parameter α :

$$(4.7) \quad Q^{\text{CSA}}(x, d, c) = \text{JSD}(J^{\mathcal{X} \rightarrow (x, d, c)} | \rho) - \text{JSD}(J^{\mathcal{X}} | \rho) - \alpha.$$

As in all top-down greedy tree induction methods, the CSA algorithm is initialised with one abstract state covering the entire feature space, $\mathcal{X} = \{\phi(\mathcal{S})\}$. It then proceeds to recursively partition the space into ever-smaller hyperrectangles (thereby growing the tree), at each step selecting x , d and c to greedily maximise Equation 4.7, i.e.

$$(4.8) \quad \operatorname{argmax}_{x \in \mathcal{X}, 1 \leq d \leq D, c \in \mathcal{C}_d} \left[Q^{\text{CSA}}(x, d, c) \right].$$

Following the notation of Section 3.2.4, \mathcal{C}_d is a finite set of candidate split thresholds along feature d , defined either manually or using the data (e.g. all midpoints between adjacent unique feature values in \mathcal{D}). As in TRIPLETREE, we grow the tree best-first (i.e. selecting the best x to split at each step) instead of the standard depth-first approach of CART. This is especially important in this context because, unlike standard impurity measures, JSD is not a local statistic of each leaf/abstract state; it depends on all other abstract states that are transitioned from and to. This non-locality property means that myopically splitting down one branch of the tree at a time is likely to yield a highly suboptimal abstraction, so we must take the more expensive approach of re-evaluating every possible split at each growth step.

The stopping criterion for terminating growth is when all possible Q^{CSA} values are ≤ 0 , which occurs earlier for larger values of the hyperparameter α . This means we can control the efficiency tradeoff between JSD and abstraction size by varying α . This is an alternative to implementing a pruning stage, which we found to provide no additional benefit in our preliminary experiments. However, some of the more sophisticated strategies considered in Section 4.7 do implement pruning in a manner that our results suggest is beneficial.

4.3.2 Contrastive Spatiotemporal Abstraction (CSTA)

Combined spatiotemporal abstraction brings additional complexity because JSD values (and their changes when new splits are added) are jointly dependent on the extant abstract states and time windows in a way that cannot be disentangled. One way to simplify this challenge is to employ a greedy two-stage algorithm.

The first step is to define an initial set of windows $\mathcal{W}_{\text{init}}$, which should be chosen to reduce the sparsity of $J^{\mathcal{X}, \mathcal{W}_{\text{init}}}$ to an acceptable level and allow the algorithm to run efficiently (note that runtime increases with $|\mathcal{W}_{\text{init}}|$). We initially hypothesise that it should otherwise have as many windows as possible to avoid excessive smoothing of the nonstationary dynamics (Section 4.7 revisits this hypothesis). If sparsity and runtime are not major issues, the ‘null’ window set $\mathcal{W}_{\text{null}} = \{[1 \leq i < 2], \dots, [k \leq i < k + 1]\}$ can be used. This initial temporal abstraction is applied to the dataset, effectively replacing the policy index i in each observation $(i, s, s') \in \mathcal{D}$ with its corresponding window index $w : l_w \leq i < u_w$. CSA is then called as a subroutine, yielding a state abstraction \mathcal{X} that maximises the JSD between the initial windows, rather than the full policy sequence.

We then discard $\mathcal{W}_{\text{init}}$ and run a secondary stage of temporal partitioning to find another set of windows \mathcal{W} that maximise JSD, while holding state abstraction \mathcal{X} constant.⁵ This is also a recursive procedure. Given an existing \mathcal{W} consisting of n windows, the window-aggregated abstract transition probabilities $J^{\mathcal{X}, \mathcal{W}}$ can be computed by Equation 4.5. Splitting the $w \in \{1, \dots, n\}$ th window at a location $i \in \{l_w + 1, \dots, u_w - 1\}$ creates two new windows with respective lower and upper boundaries at l_w, i and i, u_w respectively. Reallocating the transitions in \mathcal{D} according to the new windows in which they fall yields an enlarged transition probability array with shape $(n + 1) \times m \times m$, denoted by $J^{\mathcal{X}, \mathcal{W} \rightarrow (w, i)}$. Again, the majority of elements in this array are copied from $J^{\mathcal{X}, \mathcal{W}}$, so it can be computed efficiently.

Initiating with one window covering the entire policy sequence, $\mathcal{W} = \{[1 \leq i < k + 1]\}$, CSTA iteratively adds windows by splitting, at each step selecting w and i to greedily maximise the abstraction efficiency according to Equation 4.6, i.e.

$$(4.9) \quad \operatorname{argmax}_{1 \leq w \leq n, i \in \mathcal{C}_{\text{temporal}}} \left[Q^{\text{CSTA}}(w, i) \right],$$

where

$$(4.10) \quad Q^{\text{CSTA}}(w, i) = \text{JSD}(J^{\mathcal{X}, \mathcal{W} \rightarrow (w, i)} | \rho^{\mathcal{W} \rightarrow (w, i)}) - \text{JSD}(J^{\mathcal{X}, \mathcal{W}} | \rho^{\mathcal{W}}) - \beta,$$

and $\mathcal{C}_{\text{temporal}}$ is a finite set of candidate temporal split thresholds. In our implementation, we exclude from $\mathcal{C}_{\text{temporal}}$ any threshold that would violate a manually specified minimum window width ε . The algorithm terminates as soon as either this condition leaves no valid thresholds, or all possible Q^{CSTA} values are ≤ 0 , the latter of which occurs earlier for larger values of β .

Notice that the CSTA algorithm effectively employs the same greedy growth process twice to produce two tree structures, one representing the state abstraction \mathcal{X} , whose leaves are abstract states, and one representing the temporal abstraction \mathcal{W} , whose leaves are time windows. We have thus far avoided, and will continue to avoid, the ‘leaf’ terminology in this chapter to avoid ambiguity between these two cases.

⁵If $\mathcal{W}_{\text{init}}$ is a fine-graining of \mathcal{W} (i.e. $\forall w \in \mathcal{W}_{\text{init}}, \exists w' \in \mathcal{W} : (l_w \geq l_{w'}) \wedge (u_w \leq u_{w'})$), the final JSD is guaranteed to be upper-bounded by the value obtained after the state abstraction stage.

4.3.3 Pseudocode and Subfunction Details

Algorithm 2 contains pseudocode for CSTA. The CSA algorithm, which excludes the temporal abstraction stage, is recovered by setting $\mathcal{W}_{\text{init}} = \mathcal{W}_{\text{null}}$ and running lines 1-10 only.

Algorithm 2 Two-stage contrastive spatiotemporal abstraction for continuous spaces.

Inputs: Data \mathcal{D} , feature function ϕ , prior ρ , hyperparameters $(\mathcal{W}_{\text{init}}, \alpha, \beta, \varepsilon, \mathcal{C}, \mathcal{C}_{\text{temporal}})$
Output: State abstraction \mathcal{X} , temporal abstraction \mathcal{W}

▷ Initial data structures required for state abstraction

1: Initialise $\mathcal{X} = \text{ONESTATE}(\phi)$, $J^{\mathcal{X}, \mathcal{W}_{\text{init}}} = \text{JOINTPROBS}(\mathcal{D}, \phi, \mathcal{X}, \mathcal{W}_{\text{init}}, \rho)$

2: **while** loop not yet broken **do** ▷ State abstraction stage

▷ Try partitioning each extant abstract state at each valid split threshold

3: **for** $x \in \mathcal{X}$, $1 \leq d \leq \text{DIM}(\phi)$, $c \in \text{VALID}(\mathcal{C}_d, x)$ **do**

▷ Create expanded joint probability array for the split defined by x , d and c

4: $J^{\mathcal{X} \rightarrow (x, d, c), \mathcal{W}_{\text{init}}} = \text{SPLITSTATEPROBS}(J^{\mathcal{X}, \mathcal{W}_{\text{init}}}, x, d, c)$

5: Compute $Q^{\text{CSA}}(x, d, c)$ as in Equation 4.7 ▷ Change in α -regularised objective

6: **end for**

7: $x^*, d^*, c^* = \text{argmax}_{x, d, c} Q^{\text{CSA}}(x, d, c)$ ▷ Identify greedy split

8: **if** $Q^{\text{CSA}}(x^*, d^*, c^*) \leq 0$: **break** ▷ Break if greedy objective change is non-positive

▷ Implement selected split by updating both \mathcal{X} and $J^{\mathcal{X}, \mathcal{W}_{\text{init}}}$

9: $\mathcal{X} = \text{SPLITSTATE}(\mathcal{X}, x^*, d^*, c^*)$, $J^{\mathcal{X}, \mathcal{W}_{\text{init}}} = J^{\mathcal{X} \rightarrow (x^*, d^*, c^*), \mathcal{W}_{\text{init}}}$

10: **end while**

▷ Initial data structures required for temporal abstraction

11: Initialise $\mathcal{W} = \text{ONEWINDOW}(\mathcal{D})$, $J^{\mathcal{X}, \mathcal{W}} = \text{JOINTPROBS}(\mathcal{D}, \phi, \mathcal{X}, \mathcal{W}, \rho)$

12: **while** loop not yet broken **do** ▷ Temporal abstraction stage

▷ Try partitioning each extant time window at each valid split threshold

13: **for** $1 \leq w \leq |\mathcal{W}|$, $i \in \text{VALID}(\mathcal{C}_{\text{temporal}}, w, \varepsilon)$ **do**

▷ Create expanded joint probability array for the split defined by w and i

14: $J^{\mathcal{X}, \mathcal{W} \rightarrow (w, i)} = \text{SPLITWINDOWPROBS}(J^{\mathcal{X}, \mathcal{W}}, w, i, \rho)$

15: Compute $Q^{\text{CSTA}}(w, i)$ as in Equation 4.10 ▷ Change in β -regularised objective

16: **end for**

17: $w^*, i^* = \text{argmax}_{w, i} Q^{\text{CSTA}}(w, i)$ ▷ Identify greedy split

18: **if** $Q^{\text{CSTA}}(w^*, i^*) \leq 0$: **break** ▷ Break if greedy objective change is non-positive

▷ Implement selected split by updating both \mathcal{W} and $J^{\mathcal{X}, \mathcal{W}}$

19: $\mathcal{W} = \text{SPLITWINDOW}(\mathcal{W}, w^*, i^*)$, $J^{\mathcal{X}, \mathcal{W}} = J^{\mathcal{X}, \mathcal{W} \rightarrow (w^*, i^*)}$

20: **end while**

Subfunctions of this algorithm operate as follows:

- **ONESTATE** (line 1): Return the unit set $\{((-\infty, \infty), \dots, (-\infty, \infty))\}$, whose element is a D -tuple of identical tuples $(-\infty, \infty)$, where $D = \text{DIM}(\phi)$ (see below). This creates a single abstract state covering the entire feature space $\phi(\mathcal{S})$.
- **JOINTPROBS** (lines 1 and 11): Given a dataset \mathcal{D} and a state abstraction \mathcal{X} , use Equation 4.1 to compute $N^{\mathcal{X}}$ then normalise along the second/third dimensions to obtain joint

probabilities $J^{\mathcal{X}}$. Then, given an n -window temporal abstraction \mathcal{W} , and a k -dimensional prior vector ρ , use Equation 4.5 to compute an $n \times m \times m$ array which aggregates the values in $J^{\mathcal{X}}$ along the first dimension.

- DIM (line 3): Return D , the dimensionality of the feature space.
- VALID (lines 3 and 13): Given a set of split points and either an abstract state x or time window w , return only those split points that lie within the bounds of x or w (along feature d in the former case). In the latter case, also exclude those that lie less than a value of ε away from either bound.
- SPLITSTATEPROBS (line 4): Given an $n \times m \times m$ joint probability array $J^{\mathcal{X}, \mathcal{W}_{\text{init}}}$, return an expanded $n \times (m + 1) \times (m + 1)$ array in which the outgoing (respectively, incoming) probabilities for abstract state x are redistributed between pairs of entries on the second (third) array dimension, according to whether the feature vector for the start state $\phi(s)$ (end state $\phi(s')$) of each contained transition $(i, s, s') \in \mathcal{D}_x$ lies before or after the split point c along feature d . All other probabilities are unchanged. In our implementation, we speed up this subfunction by maintaining a temporary expanded array for each x, d pair, which is updated incrementally during a sweep over sorted split points c .
- SPLITSTATE (line 9): Given a state abstraction \mathcal{X} , return a copy with the tuple for abstract state x , denoted by $((l_{x,1}, u_{x,1}), \dots, (l_{x,D}, u_{x,D}))$, removed. It is replaced by two new tuples which are copies of the removed one except for their d th elements, which become $(l_{x,d}, c)$ and $(c, u_{x,d})$ respectively.
- ONEWINDOW (line 11): Return the unit set $\{(l_1, u_1)\}$, where $l_1 = 1$, $u_1 = k + 1$, and k is the number of policies represented in \mathcal{D} . This creates a single time window covering the entire policy sequence.
- SPLITWINDOWPROBS (line 14): Given an $n \times m \times m$ joint probability array $J^{\mathcal{X}, \mathcal{W}}$, return an expanded $(n + 1) \times m \times m$ array in which the probabilities for time window w are redistributed between pairs of entries on the first array dimension, according to whether the policy index j of each contained transition $(j, s, s') \in \mathcal{D} : l_w \leq j \leq u_w$ lies before or after the temporal split point i . The redistribution is also weighted by the prior ρ . All other probabilities are unchanged. Our implementation of this function also uses incrementally updated temporary arrays to improve speed.
- SPLITWINDOW (line 19): Given a temporal abstraction \mathcal{W} , return a copy with the tuple for time window w removed and replaced by new two tuples (l_w, i) , (i, u_w) .

4.4 Scaling of Jensen-Shannon Divergence with m and n

In this section, we seek to gain some understanding of the expected behaviour of the Jensen-Shannon divergence as we vary both the number of abstract states m and the number of time windows n . We do this by both theoretical and empirical means.

The analysis in this section is not a prerequisite for understanding the main experiments in Sections 4.5 and 4.6. The reader may wish to skip ahead to those sections for a general empirical validation of our method, before returning here for deeper theoretical understanding.

4.4.1 Theoretical Analysis for m

For reference, the form of the Jensen-Shannon divergence given in Equation 4.3 is

$$\text{JSD}(J^{\mathcal{X}}|\boldsymbol{\rho}) = -\sum_{i=1}^k \rho_i \mathcal{H}(J_i^{\mathcal{X}}) + \mathcal{H}\left(\sum_{i=1}^k \rho_i J_i^{\mathcal{X}}\right),$$

which can be expanded as follows:

$$\begin{aligned} \text{JSD}(J^{\mathcal{X}}|\boldsymbol{\rho}) &= \sum_{x_p \in \mathcal{X}} \sum_{x_q \in \mathcal{X}} z_{p,q}, \\ \text{where } z_{p,q} &= \left(\sum_{i=1}^k \rho_i J_{i,x_p,x_q} \log J_{i,x_p,x_q} \right) - \left(\sum_{i=1}^k \rho_i J_{i,x_p,x_q} \right) \log \left(\sum_{i=1}^k \rho_i J_{i,x_p,x_q} \right). \end{aligned}$$

Consider the scaling of JSD with m under a binary split operation, which involves partitioning one abstract state $x_0 \in \mathcal{X}$ into two new ones x_1 and x_2 . This analysis is of interest because our practical tree-based CSA algorithm employs a binary splitting approach. The following conservation equations apply to the abstract transition probabilities of each policy $i \in \{1, \dots, k\}$:

$$\begin{aligned} J_{i,x_0,x_p} &= J_{i,x_1,x_p} + J_{i,x_2,x_p}, \quad \forall x_p \in \mathcal{X}^- \quad (\text{Transitions out of } x_0 \text{ are conserved}); \\ J_{i,x_p,x_0} &= J_{i,x_p,x_1} + J_{i,x_p,x_2}, \quad \forall x_p \in \mathcal{X}^- \quad (\text{Transitions into } x_0 \text{ are conserved}); \\ J_{i,x_0,x_0} &= J_{i,x_1,x_1} + J_{i,x_1,x_2} + J_{i,x_2,x_1} + J_{i,x_2,x_2} \quad (\text{Transitions within } x_0 \text{ are conserved}), \end{aligned}$$

where $\mathcal{X}^- = \mathcal{X} \setminus \{x_0\}$, and we have omitted the superscripted \mathcal{X} from $J_{i,\cdot,\cdot}$ to reduce visual clutter. All other transition probabilities (i.e. between the members of \mathcal{X}^-) remain unchanged.

Let δ_{JSD} denote the difference between the JSD values before and after this split is made. Most terms of this difference cancel, and we are left with

$$\delta_{\text{JSD}} = \sum_{x_p \in \mathcal{X}^-} \left(z_{p,1} + z_{1,p} + z_{p,2} + z_{2,p} - z_{p,0} - z_{0,p} \right) + z_{1,1} + z_{1,2} + z_{2,1} + z_{2,2} - z_{0,0}.$$

A general analysis of δ_{JSD} is not straightforward, so here we only consider the special case of $k = 2$ policies with prior weighting $\rho_1 = \rho_2 = \frac{1}{2}$. As a lower bound, if the underlying state transition probabilities in \mathcal{S} are identical for policies 1 and 2, it is easy to show that all terms cancel and $\delta_{\text{JSD}} = 0$. More interestingly, we can upper bound the JSD difference by considering

the ‘perfect splitting’ case when policies 1 and 2 have the *same* inbound and outbound transition probabilities for x_0 (i.e. $J_{1,x_p,x_0} = J_{2,x_p,x_0}$ and $J_{1,x_0,x_p} = J_{2,x_0,x_p}$, $\forall x_p \in \mathcal{X}^- \cup \{x_0\}$), but policy 1 *only* visits x_1 and policy 2 *only* visits x_2 . This leads to the following simplification of the terms $z_{p,1} + z_{p,2} - z_{p,0}$ for each $x_p \in \mathcal{X}^-$:

$$\begin{aligned} z_{p,1} + z_{p,2} &= 2 \left(\frac{1}{2} J_{\cdot,x_p,x_0} \log J_{\cdot,x_p,x_0} - \frac{1}{2} J_{\cdot,x_p,x_0} \log \frac{1}{2} J_{\cdot,x_p,x_0} \right); \\ z_{p,0} &= \frac{2}{2} J_{\cdot,x_p,x_0} \log J_{\cdot,x_p,x_0} - \frac{2}{2} J_{\cdot,x_p,x_0} \log \frac{2}{2} J_{\cdot,x_p,x_0} = 0; \\ z_{p,1} + z_{p,2} - z_{p,0} &= J_{\cdot,x_p,x_0} \log \frac{J_{\cdot,x_p,x_0}}{\frac{1}{2} J_{\cdot,x_p,x_0}} - 0 = J_{\cdot,x_p,x_0} \log 2, \end{aligned}$$

where the subscripted placeholder “ \cdot ” indicates that this joint probability is the same for both policies. An equivalent simplification can be derived for $z_{1,p} + z_{2,p} - z_{0,p}$, and also for $z_{1,1} + z_{2,2} - z_{0,0}$. We also know that $z_{1,2} = z_{2,1} = 0$, because (by assumption) neither of the two policies ever visits both x_1 and x_2 . Substituting back into the equation for δ_{JSD} gives

$$\delta_{\text{JSD}} = \log 2 \sum_{x_p \in \mathcal{X}^-} \left(J_{\cdot,x_p,x_0} + J_{\cdot,x_0,x_p} \right) + J_{\cdot,x_0,x_0} = \log 2 \left(M_{\cdot,x_0} + \sum_{x_p \in \mathcal{X}^-} J_{\cdot,x_p,x_0} \right),$$

where $M_{\cdot,x_0} = \sum_{x_p \in \mathcal{X}} J_{\cdot,x_0,x_p}$ is the marginal visitation probability for x_0 , obtained by summing over all joint outbound transition probabilities. This quantity is the same for both policies.

What can we say about the behaviour of δ_{JSD} in expectation? For any policy i , $\sum_{x \in \mathcal{X}} M_{i,x} = 1$, so if x_0 was selected for splitting with uniform random probability, $\mathbb{E}_{x_0}[M_{\cdot,x_0}] = \frac{1}{m}$ (where m is the number of abstract states *before* the split is made). We can also bring the expectation of the second term inside the sum:

$$\mathbb{E}_{x_0}[\delta_{\text{JSD}}] = \log 2 \left(\mathbb{E}_{x_0}[M_{\cdot,x_0}] + \sum_{x_p \in \mathcal{X}^-} \mathbb{E}_{x_0}[J_{\cdot,x_p,x_0}] \right) = \log 2 \left(\frac{1}{m} + \sum_{x_p \in \mathcal{X}^-} \mathbb{E}_{x_0}[J_{\cdot,x_p,x_0}] \right).$$

We cannot say much more about the second term in general, aside from that it will depend on the amount of ‘inertia’ in the Markov chain for each policy: the proportion of timesteps in which it remains in the same abstract state as opposed to transitioning elsewhere. If inertia is very high, the first term of our expression for $\mathbb{E}_{x_0}[\delta_{\text{JSD}}]$ will dominate, in which case

$$\mathbb{E}_{x_0}[\delta_{\text{JSD}}] = \frac{\log 2}{m} \implies \mathbb{E}[\text{JSD}] = \log 2 \log m,$$

since $\frac{d}{dy}(a \log y) = \frac{a}{y}$. This result suggests that in the high-inertia context, JSD is approximately logarithmic in m . Finally, as a looser upper bound, we can say for certain that $J_{\cdot,x_p,x_0} \leq M_{\cdot,x_p}$, so $\mathbb{E}_{x_0}[J_{\cdot,x_p,x_0}] \leq \frac{1}{m}$, $\forall x_p \in \mathcal{X}^-$. Therefore,

$$\mathbb{E}_{x_0}[\delta_{\text{JSD}}] \leq \log 2 \left(\frac{1}{m} + \sum_{x_p \in \mathcal{X}^-} \frac{1}{m} \right) = \log 2 \left(\frac{1}{m} + \frac{m-1}{m} \right) = \log 2 \implies \mathbb{E}[\text{JSD}] \leq m \log 2.$$

since $\frac{d}{dy}(ay) = a$. In this (extremely generous) upper bound case, JSD is linear in m . As this case will be realised very rarely in practice, it is reasonable to say that the scaling will almost always be sublinear, and more tentatively, that it is approximately logarithmic.

4.4.2 Theoretical Analysis for n

A more general analysis is possible for n . When temporal abstraction is applied to aggregate data from k sequentially ordered policies into n time windows, the Jensen-Shannon divergence between those windows is given by

$$\begin{aligned} \text{JSD}(J^{\mathcal{X},\mathcal{W}}|\rho^{\mathcal{W}}) &= \left(\sum_{w=1}^n \rho_w^{\mathcal{W}} \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} J_{w,x,x'}^{\mathcal{X},\mathcal{W}} \log J_{w,x,x'}^{\mathcal{X},\mathcal{W}} \right) \\ &\quad - \left(\sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} \left(\sum_{w=1}^n \rho_w^{\mathcal{W}} J_{w,x,x'}^{\mathcal{X},\mathcal{W}} \right) \log \left(\sum_{v=1}^n \rho_v^{\mathcal{W}} J_{v,x,x'}^{\mathcal{X},\mathcal{W}} \right) \right). \end{aligned}$$

Consider splitting one time window $w_0 \in \{1, \dots, n\}$ into two new ones w_1 and w_2 , as is done in our practical CSTA algorithm. Under such a binary split operation, the following conservation equations hold for all $x \in \mathcal{X}, x' \in \mathcal{X}$:

$$\begin{aligned} \rho_{w_0} &= \rho_{w_1} + \rho_{w_2} \quad (\text{Sum of prior weights are conserved}); \\ \rho_{w_0} J_{w_0,x,x'} &= \rho_{w_1} J_{w_1,x,x'} + \rho_{w_2} J_{w_2,x,x'} \quad (\text{Joint probabilities are prior-weighted average}), \end{aligned}$$

where the superscripted \mathcal{X} and \mathcal{W} are omitted from both $\rho^{\mathcal{W}}$ and $J_{\cdot,\cdot,\cdot}^{\mathcal{X},\mathcal{W}}$ to reduce visual clutter.

Again taking δ_{JSD} to be the difference between the JSD values before and after the split, all of the second term (of the form given above), and most components of the first term, cancel out. We are left with

$$\delta_{\text{JSD}} = \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} \left(\rho_{w_1} J_{w_1,x,x'} \log J_{w_1,x,x'} + \rho_{w_2} J_{w_2,x,x'} \log J_{w_2,x,x'} - \rho_{w_0} J_{w_0,x,x'} \log J_{w_0,x,x'} \right).$$

Applying the conservation equations and multiplying and dividing by ρ_{w_0} , we can rewrite as

$$\begin{aligned} \delta_{\text{JSD}} &= \rho_{w_0} \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} \left(\frac{\rho_{w_1}}{\rho_{w_0}} J_{w_1,x,x'} \log J_{w_1,x,x'} + \frac{\rho_{w_2}}{\rho_{w_0}} J_{w_2,x,x'} \log J_{w_2,x,x'} \right. \\ &\quad \left. - \left(\frac{\rho_{w_1}}{\rho_{w_0}} J_{w_1,x,x'} + \frac{\rho_{w_2}}{\rho_{w_0}} J_{w_2,x,x'} \right) \log \left(\frac{\rho_{w_1}}{\rho_{w_0}} J_{w_1,x,x'} + \frac{\rho_{w_2}}{\rho_{w_0}} J_{w_2,x,x'} \right) \right). \end{aligned}$$

By inspection, everything aside from the leading ρ_{w_0} exactly matches the definition of the JSD itself, calculated pairwise between windows w_1 and w_2 , and using renormalised weights:

$$\delta_{\text{JSD}} = \rho_{w_0} \text{JSD} \left(\left[J_{w_1}, J_{w_2} \right] \middle| \left[\frac{\rho_{w_1}}{\rho_{w_0}}, \frac{\rho_{w_2}}{\rho_{w_0}} \right] \right).$$

δ_{JSD} is lower-bounded by zero in the case when w_1 and w_2 have identical transition probabilities in \mathcal{X} (since pairwise JSD will be zero). As an upper bound, we again consider a ‘perfect splitting’ case where the two new windows have *no* abstract state transitions in common (i.e. $J_{w_1,x,x'} > 0 \implies J_{w_2,x,x'} = 0$ and $J_{w_2,x,x'} > 0 \implies J_{w_1,x,x'} = 0, \forall x \in \mathcal{X}, x' \in \mathcal{X}$),

and equal prior weighting $\rho_{w_1} = \rho_{w_2} = \frac{\rho_{w_0}}{2}$. Returning to the expanded form of δ_{JSD} and substituting in these relations, we obtain

$$\begin{aligned} \delta_{\text{JSD}} &= \rho_{w_0} \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} \left(\frac{1}{2} J_{w_1 \vee w_2, x, x'} \log J_{w_1 \vee w_2, x, x'} - \left(\frac{1}{2} J_{w_1 \vee w_2, x, x'} \right) \log \left(\frac{1}{2} J_{w_1 \vee w_2, x, x'} \right) \right) \\ &= \frac{\rho_{w_0}}{2} \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} J_{w_1 \vee w_2, x, x'} \log \frac{J_{w_1 \vee w_2, x, x'}}{\frac{1}{2} J_{w_1 \vee w_2, x, x'}} = \frac{\rho_{w_0} \log 2}{2} \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} J_{w_1 \vee w_2, x, x'} \xrightarrow{2} = \rho_{w_0} \log 2, \end{aligned}$$

where the subscript “ $w_1 \vee w_2$ ” stands for *either* w_1 or w_2 ; whichever of the two has nonzero transition probability for each x, x' pair (it is never both). In the final line above, the double sum cancels to 2 because for each of the two windows, the sum of joint probabilities is 1.

Finally, we know that $\sum_{w=1}^n \rho_w = 1$, so if we assume that w_0 was selected uniform-randomly, $\mathbb{E}_{w_0}[\rho_{w_0}] = \frac{1}{n}$ (where n is the number of policies *before* the split is made). Therefore,

$$\mathbb{E}_{x_0}[\delta_{\text{JSD}}] = \frac{\log 2}{n} \implies \mathbb{E}[\text{JSD}] = \log 2 \log n,$$

since $\frac{d}{dy}(a \log y) = \frac{a}{y}$. We can thus conclude from this upper-bound analysis that JSD is reliably sublinear in n .

4.4.3 Empirical Validation

As a simple empirical validation of the preceding results, we generate datasets of $k = 100$ random trajectories of length $H = 100$ in the 2D region $\mathcal{S} = [0, 1]^2$ according to a Gaussian random walk model, parameterised by $i \in \{1, \dots, k\}$:

$$s_{i,0} \sim \text{uniform}(\mathcal{S}); \quad s_{i,t} = \min(\max(s_{i,t-1} + \eta, 0), 1), \quad \forall t \in \{1, \dots, H-1\},$$

$$\text{where } \eta \sim \mathcal{N} \left(\begin{bmatrix} v \sin(\frac{3\pi i}{2k}) & v \cos(\frac{3\pi i}{2k}) \end{bmatrix}, \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix} \right).$$

In each trajectory, Gaussian noise with standard deviation σ is added to a mean velocity vector with magnitude v , which is gradually rotated through 270° over the sequence $i \in \{1, \dots, k\}$. Hence, each trajectory is a sample from a different Markov chain, with ones close together in the sequence tending to be more similar than those far apart. We generate four datasets with various values of v and σ .

For each dataset, we perform top-down tree abstraction of the region up to a maximum size $m = 100$ and measure the resultant JSD. For some values of m , we then go on to perform temporal abstraction up to a maximum of $n = 100$ windows. To select the next split to make at each point, we use both a random strategy and the CSTA algorithm presented in Section 4.3.2, which greedily maximises the JSD gain on each step. As Figure 4.4 shows, the scaling of JSD is consistently sublinear in both m and n , which matches our theoretical predictions. It is also always higher when the CSTA algorithm is used, compared with random splitting. Note that these datasets have an extremely simple dynamical structure; the advantage of principled split selection using CSTA is likely to be far greater in practical applications.

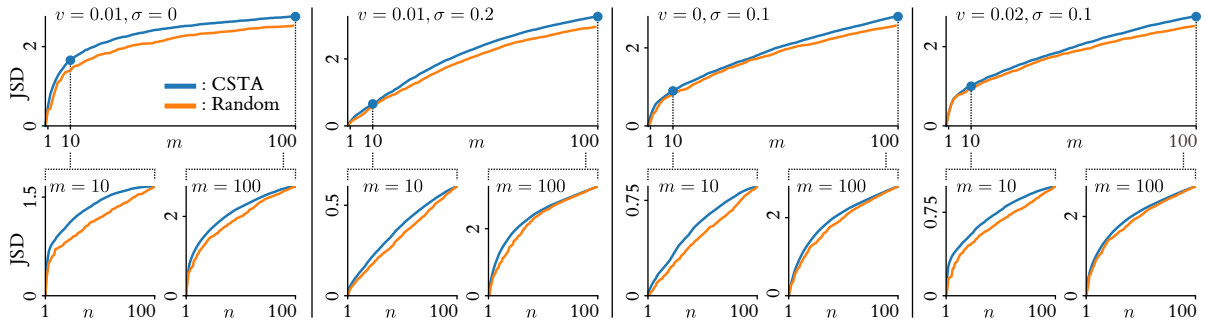


Figure 4.4: Scaling of JSD with m and n for random trajectories in $[0, 1]^2$ with various velocity and noise parameters v, σ , and when selecting both random axis-aligned splits (orange) and those that maximise the contrastive abstraction objective (blue). Top row: scaling with m . Bottom row: scaling with n for the specific values of m shown.

4.5 2D Maze Experiment

In the following two sections, we apply our method to construct interpretable contrastive summaries of the learning of RL agents in two episodic environments⁶ with real vector state representations $\phi(\mathcal{S}) = \mathbb{R}^D$.

First, we consider a simple continuous navigation task, which is depicted in Figure 4.5 (a). The state features are the agent’s horizontal and vertical positions in a bounded 2D arena, $pos_x, pos_y \in [0, 10]^2$. The objective is to move to a goal region ($pos_x \geq 8 \wedge pos_y \geq 7$, shown in green) without entering a penalty region ($pos_x \geq 8 \wedge 3 \leq pos_y < 7$, shown in red), and while navigating around a pair of horizontal walls at $pos_y = 3$ and $pos_y = 7$, which block transverse motion. At the start of an episode, the agent is initialised in a random position (but never in the goal or penalty regions). At each timestep, the agent’s action specifies its horizontal and vertical velocities $vel_x, vel_y \in [-0.25, 0.25]^2$, which are clipped if the resultant motion vector would intersect a wall or external arena boundary. The agent is given a reward of +100 for entering the goal region, and -100 for entering the penalty region. The episode is also terminated immediately if either the goal or penalty region is entered, or after 200 timesteps.

We train a soft actor-critic (SAC) [107] RL agent to solve this task.⁷ In a minor departure from convention, we modify the SAC algorithm to perform policy updates on the final timestep of each episode only, thereby guaranteeing that the policy (and thus, the induced Markov chain) is stationary within each episode. This allows us to avoid the theoretical complication of the policy index i changing midway through an episode, thereby making the observed transitions from policy π_i dependent on the dynamics of the previous policy π_{i-1} .

⁶As in TRIPLETREE, we can handle episodic endings by adding a ‘pseudo-state’ \emptyset which is transitioned to when an episode terminates. This requires a very minor modification of the equations given up to this point.

⁷Agent hyperparameters are as follows. We use a discount factor of $\gamma = 0.99$, an entropy regularisation coefficient of $\alpha = 0.2$, a replay buffer capacity of 20000 samples and a minibatch size of 64. All networks (policy and value functions) have two hidden layers of 256 units each and are trained by backpropagation using the Adam optimiser [142]. The learning rate is $1e^{-4}$ for the policy network and $1e^{-3}$ for the value network, and lagging target network parameters are updated by Polyak averaging with an interpolation factor of 0.995.

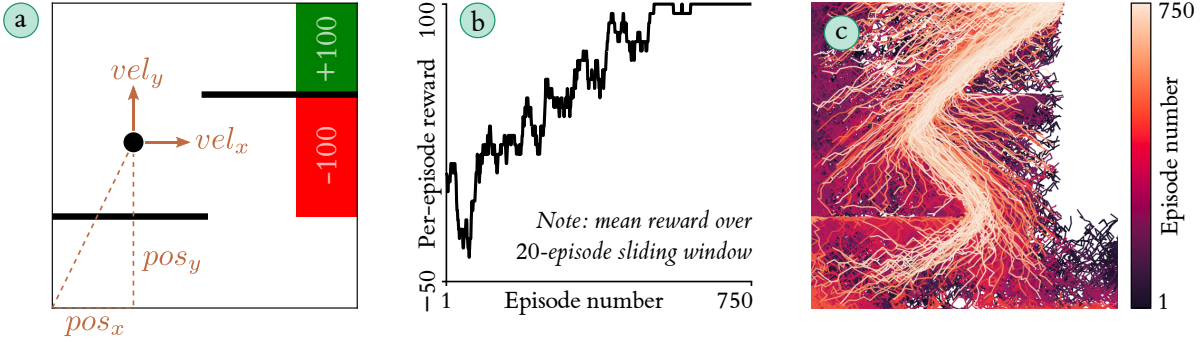


Figure 4.5: 2D Maze environment, learning curve and transition data for 750 learning episodes.

Figure 4.5 (b) shows the agent’s *learning curve*, which is a 20-episode moving average of the reward obtained as it makes $k = 750$ once-per-episode policy updates. By the end of learning, it is consistently able to reach the goal and attain the maximum reward of 100. In (c), we plot all transition data generated by the agent during its learning, coloured by episode/policy index, which shows how the policy converges to a ‘zigzag’ behaviour that moves to the goal while avoiding both the walls and the penalty region. Collectively, these transitions form the dataset \mathcal{D} that we use to learn a contrastive spatiotemporal abstraction.

4.5.1 State Abstraction Process

We begin by setting $\mathcal{W}_{\text{init}} = \mathcal{W}_{\text{null}}$ (i.e. initially allocating every episode to a separate window), the prior ρ to be proportional to the length of each episode (so that episodes that are terminated early do not have an outsized impact on calculated transition probabilities) and the candidate split thresholds \mathcal{C} at uniform intervals of 0.1 along both pos_x and pos_y . We then commence the state abstraction process, which is visualised in Figure 4.6.

The two subplots (a) and (b) show the JSD that would result for every candidate threshold along pos_x and pos_y respectively for the abstract state that is chosen for splitting at each step. The split is made at the threshold that maximises JSD. For instance, the blue curves correspond to the very first split decision after the tree is initialised with a single leaf. A sharp peak is visible at $pos_y = 3.0$ (solid curve in (b)), and no equivalently high peak exists along pos_x (dashed curve in (a)), so $pos_y = 3.0$ is selected as the split threshold. Curves for the second split are shown in orange. In this case, $pos_x = 3.3$ is selected. Subsequent splits are guaranteed to increase JSD (notice how the curves are stacked along the JSD axis), but the incremental gain from each successive split tends to diminish as predicted by our analysis in Section 4.4.1. This is also visible in the plot of JSD as a function of abstraction size m , shown in (c), which has a similar profile to those in Figure 4.4. With the regularisation hyperparameter set to $\alpha = 0.05$, the objective in Equation 4.4 (JSD minus regularisation) is maximised when $m = 12$, and the abstraction process terminates. The resultant set of rectangular abstract states is shown in (d), with boundaries coloured to match the corresponding JSD curves.

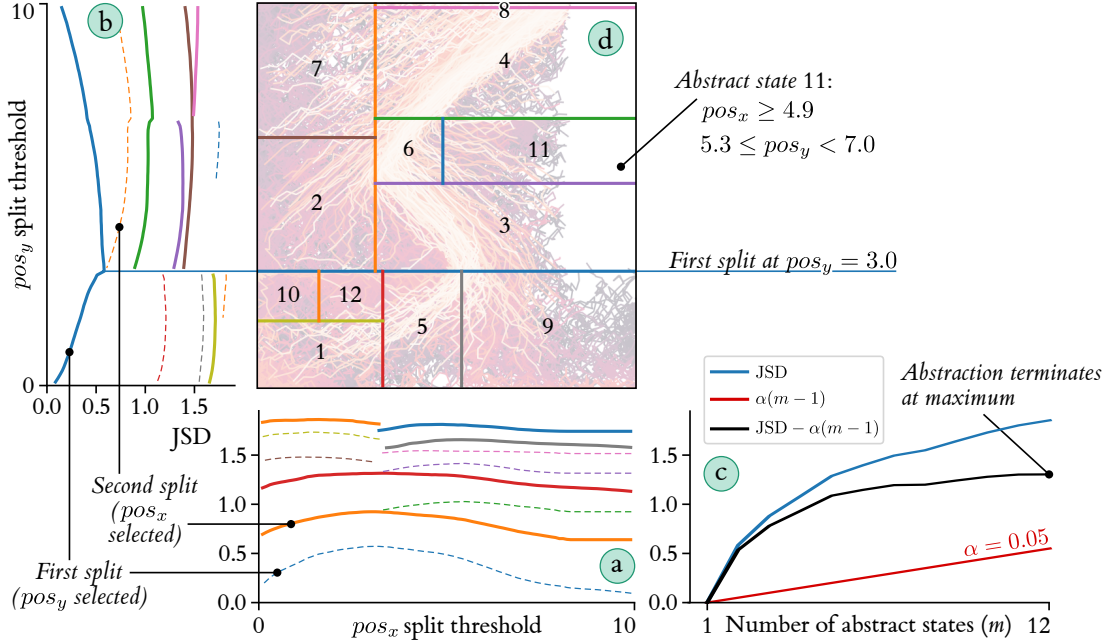


Figure 4.6: Visualisation of state abstraction process for 2D Maze environment.

To reflect on this process, we can see that our algorithm makes its first and third splits at $pos_y = 3$ and $pos_y = 7$ respectively, which align exactly with the locations of the two horizontal walls. The sharp peaks in JSD around these locations, visible in subplot (b), suggest that this is no coincidence, and that the changing ability of the agent to pass each wall (i.e. transition across these abstract state boundaries) is a major source of divergence among the sequence of policies exhibited by the agent during its learning. Other splits have similarly intuitive origins, with abstract states 6 and 8 respectively covering a critical junction at the upper wall where the agent has to learn a sharp right turn (taking it to abstract state 4), and part of the upper arena boundary along which the agent learns to ‘slide’ to the goal in the latter stages of learning. These observations provide evidence that the JSD-based split criterion can identify dynamically important thresholds in the state feature space.

4.5.2 Temporal Abstraction Process

As outlined in Section 4.3.2, we then discard $\mathcal{W}_{\text{init}}$ and perform temporal abstraction while holding the state abstraction fixed. We use the exhaustive set $\{2, \dots, 750\}$ as the candidate split thresholds $\mathcal{C}_{\text{temporal}}$ and $\varepsilon = 25$ as the minimum window width. The temporal abstraction process is visualised in Figure 4.7. As above, the lowest (blue) curve in subplot (a) shows the JSD that would result from placing the first split at each possible temporal threshold. In this case, it is maximised at episode/policy index $i = 397$, so a split is made at this point to partition the learning sequence into two windows. Subsequent splits are again guaranteed to increase JSD (note stacking of curves), but gains tend to diminish as predicted by the theory in Section 4.4.1. Subplot (b) shows that with our choice of regularisation hyperparameter $\beta = 0.01$, the objective

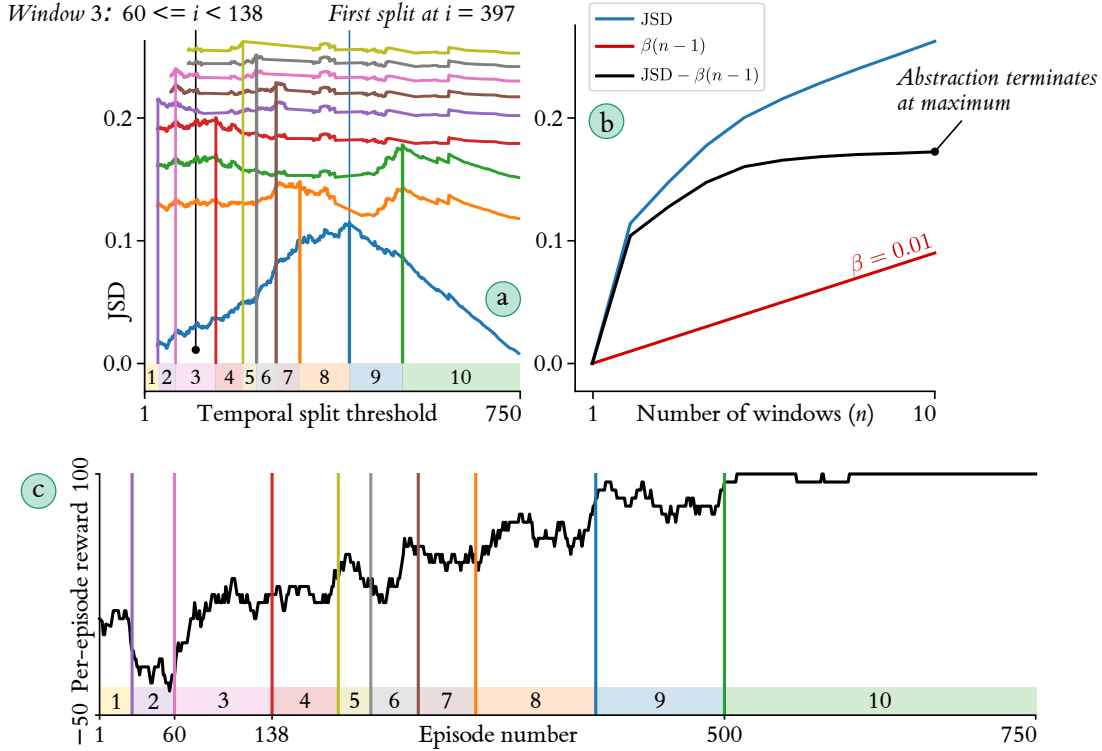


Figure 4.7: Visualisation of temporal abstraction process for 2D Maze environment.

in Equation 4.6 is maximised at $n = 10$ time windows, and the process terminates at this point.

We have seen how contrastive state abstraction can find critical locations in the state space around which the agent’s dynamics vary significantly during learning, namely the walls in the maze. As an analogous validation of the temporal abstraction, (c) overlays the 10 time windows onto the agent’s learning curve (copied from Figure 4.5 (b)). Notice the alignment between the window boundaries and various jumps and directional changes in reward. This provides good evidence that the JSD-based split criterion is effective for partitioning learning into periods of similar behaviour. The alignment of windows with learning progress also imbues them with additional semantics. For example, window 2 can be understood as “a period of temporary worse-than-initial performance” (the reason for which is identified in Section 4.5.4) and the final window 10 as “the agent’s behaviour once learning has converged”. However, some window boundaries (e.g. between 3 and 4) do not correspond to reward change points. Such boundaries reflect changes in agent dynamics that do not alter the net probability of entering the goal or penalty regions, but may still be important for understanding learning trends.

4.5.3 Pairwise Window Divergence

Although our abstraction approach computes (and maximises) overall JSD among the entire set of windows, it can also be computed pairwise, yielding the matrix in Figure 4.8 (a). This matrix provides a reward-independent representation of the nonstationary dynamics, which

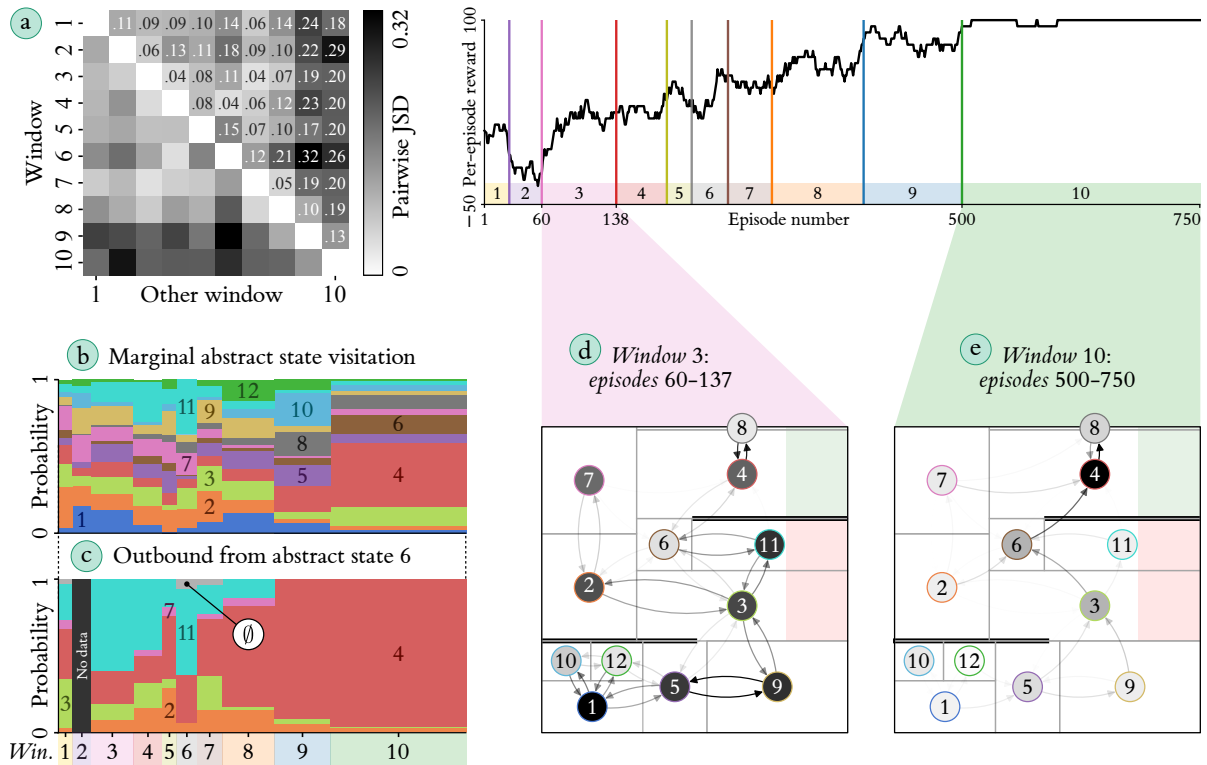


Figure 4.8: Additional results figures for 2D Maze environment. Note: \emptyset = episode termination.

is complementary to the learning curve and can be used to identify trends and outliers for follow-up investigation. The complexity of practical reinforcement learning is evident here: although the window pairs that are furthest apart in time tend to be the most divergent, this trend is far from monotonic, and neighbouring windows can be highly dissimilar (e.g. 5 and 6), suggesting rapid dynamical changes. A notable result is the dark banding for the final two windows 9 and 10, indicating that both are dissimilar to all earlier windows. Recall from Figure 4.7 that the first step of temporal abstraction was to partition these two windows off from the rest by splitting at episode 397. This finding reinforces that this time point was especially critical to the agent’s convergence to high-reward behaviour. It also suggests that (as in standard tree induction) the order in which the CSTA algorithm makes splits carries valuable information about similarities. Another outlier is window 6, which has a visually unusual JSD pattern compared with those on either side. We uncover the origin of this anomaly below.

4.5.4 Visitation and Transition Time Series

The stacked bar charts in Figure 4.8 (b) and (c) provide more explicit insight into the changing dynamics across all time windows. In (b), the bar heights reflect the marginal visitation probability to each abstract state in each window, and in (c), they represent the conditional outbound probabilities from abstract state 6 only, excluding self-transitions from $6 \rightarrow 6$. Note

that window 2 is labelled “No data” because the agent never visits abstract state 6 in that period. In these visualisations, we can again see the non-monotonicity of the RL process, as probabilities increase and decrease repeatedly over time. Trends that occur during the agent’s learning include the following:

- Marginal abstract state visitation is initially rather uniform but eventually skews towards abstract state 4 (which contains the goal) and away from 2, 7 and 9 (which lie outside of the zigzag path that marks the most efficient route to the goal). Visits to state 3 (which must be traversed to reach the goal) remain largely unchanged throughout learning.
- Temporary learning aberrations include two ‘waves’ of visitation to state 1, in which the agent fails to exit the bottom-left corner. The peaks of these waves occur in window 2 (which may explain the markedly lower reward than window 1) and window 8.
- From window 8 onwards, the outbound $6 \rightarrow 4$ transition comes to dominate the conditional probabilities. It appears that this time threshold is when the agent learns to reliably turn right past the upper wall rather than continue straight or turn back.
- The $6 \rightarrow 11$ transition briefly spikes in window 6, and as a result, the agent becomes far more likely to get stuck below the wall in state 11. This explains why window 6 is identified as anomalous in the pairwise JSD matrix.

4.5.5 Transition Graph Comparison

We examine the abstract state dynamics of windows 3 and 10 in detail using the transition graphs in Figure 4.8 (d) and (e), where node and edge opacities scale with abstract state visitation and joint transition probabilities respectively. They provide further contrastive insight:

- The sparser and less symmetric graph for window 10 indicates a general trend towards less random and more goal-directed behaviour.
- Transitions providing progress to the goal (e.g. $3 \rightarrow 6$) become more common, while those leading to corners or dead-ends (e.g. $9 \rightarrow 5$) occur less frequently.
- Abstract state 2 sees both a decrease in visitation and a focusing of outbound transitions to abstract state 6 only. The agent has no incentive to visit this abstract state as it does lie on the efficient zigzag path, but on the rare occasions that it does (due to policy stochasticity), it tends to transition back onto the path.
- Inbound transitions to abstract state 1 disappear entirely. In window 10, the agent only visits the state if initialised there at the start of an episode.

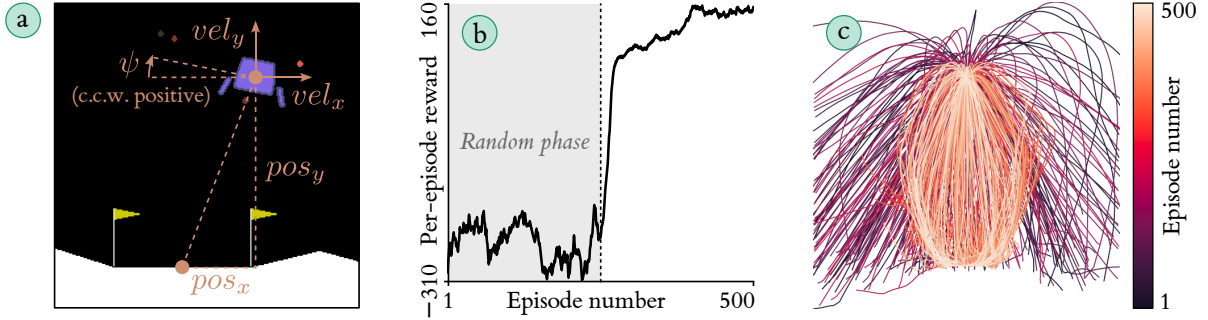


Figure 4.9: LunarLander, learning curve and transition data for 500 random/learning episodes.

4.6 LunarLander Experiment

The preceding results validate that contrastive spatiotemporal abstraction enables insight into the dynamics of agent learning beyond that provided by simple performance metrics such as a learning curve, which would otherwise require an exhaustive review of transition data. However, in the simple maze domain, this insight is somewhat tautological (i.e. “the agent reaches the goal by moving to the goal”). For this reason, we now consider a more high-dimensional environment where direct visualisation of the state abstraction geometry is impossible.

Echoing the previous chapter, we turn to LunarLander to provide this more challenging evaluation. Recall that in this environment, shown in Figure 4.9 (a), the agent controls two engines to land an aerial craft on a pad, with ± 100 reward for landing/crashing, and shaping reward promoting smooth flight. For further details, refer back to Section 3.10. We train a SAC RL agent to solve this task, using the same training setup as in the 2D Maze experiment aside from a minor adjustment of hyperparameters.⁸ However, we first force the agent to execute 250 episodes of random behaviour before enabling the SAC algorithm for a further 250 (so $k = 500$). This enables a sense-check of the temporal abstraction, which is described below.

Figure 4.9 (b) shows that the initial random policy receives strongly negative reward. Performance increases rapidly as soon as learning is enabled, eventually converging to a positive reward of ≈ 150 . The transition data plot (c) (in pos_x, pos_y coordinates) shows how the lander’s descent becomes more controlled over time, narrowing to a stable landing envelope.

4.6.1 Abstraction Structure

Running the CSTA algorithm⁹ yields $m = 12$ abstract states, which are displayed as a tree diagram in Figure 4.10 (a). The inherent symmetry of the environment is reflected in splits along the horizontal position feature pos_x , with the first very close to the centreline (thereby partitioning the space into left and right halves) and others near ± 0.1 and ± 0.2 (note that

⁸Specifically, we reduce the learning rate for policy and value network updates to $5e^{-5}$ and $5e^{-4}$ respectively, and reduce the target network Polyak interpolation factor to 0.99.

⁹Hyperparameter values: prior ρ proportional to episode lengths; $\mathcal{W}_{\text{init}} = \mathcal{W}_{\text{null}}$; $\alpha = 0.05$; $\beta = 0.01$; $\varepsilon = 15$; \mathcal{C}_d defined as percentiles of data along each feature d ; $\mathcal{C}_{\text{temporal}}$ defined as exhaustive set $\{2, \dots, 500\}$.

the landing pad extends to ± 0.2). 8 out of 11 splits concern the agent’s horizontal dynamics (including velocity vel_x), which suggests that most temporal contrasts occur along this direction. Vertical dynamics are largely gravity-driven, so it makes sense that they remain more invariant over learning. Exceptions are the splits along pos_y to create abstract states 2 and 5, which correspond to being low and central (i.e. correct landing) on the right and left respectively. The tree’s rule structure allows us to find simple correspondences of this form for all 12 abstract states, which are colour-coded in a *semantic key* (b). Although some manual effort and domain knowledge is required to construct this key, it provides an additional layer of abstraction that simplifies the discussion of high-level dynamics and trends. The assignment of shorthand natural language labels to abstract states is similar to work by McCalzone et al. [171], who enable this by performing state abstraction using Boolean classifiers based on user-specified predicates.

The temporal abstraction stage yields $n = 6$ time windows, which are shown overlaid onto the agent’s learning curve in (c). All splits are made after the random phase (i.e. during learning) and before the agent converges to a stable reward of around 150 per episode. This passes the aforementioned sense-check, and aligns with our theoretical understanding: JSD is maximised by splitting at times when transition probabilities are changing, and splitting a window containing unchanging (random) behaviour should give zero JSD gain in expectation. As in the 2D Maze experiment, the final window begins at the point where the learning curve becomes horizontal, so that it effectively represents the agent’s behaviour at convergence.

4.6.2 Visitation Time Series

Figure 4.10 (d) shows the agent’s abstract state visitation over time, which we can interpret by referring to the semantic key. In the random phase (window 1) the majority of time is spent “drifting out” from the centre (states 3, 4) but this is rapidly unlearned. Visits to the two “landing” states 2 and 5 increase near-monotonically, while others oscillate in prevalence before convergence. The two “wide” states 1 and 9 settle to approximately equal visitation (indicating a kind of symmetry) after reaching a higher peak in window 4.

4.6.3 Transition Graph Comparison

A richer narrative emerges from the transition graphs for all six windows in Figure 4.10 (e). For the sake of brevity, we focus on states in the left half of the state space, enclosed in dotted lines. The trends are complex and nonlinear, as expected from a stochastic learning agent, but can be interpreted with the aid of the tree diagram and semantic key. Our annotations tell the story of over-rotated leftward motion in state 3 giving way to a slower leftward drift $10 \rightarrow 7 \rightarrow 1$, punctuated by a brief abandonment of left-side approaches, and finally a convergence to landing state 5 with a preference for arced approaches via states 1 and 7. For more details on these trends, refer to the annotations themselves. We reflect again that such insight would typically require a laborious review of the learning history, but is made accessible by contrastive abstraction.

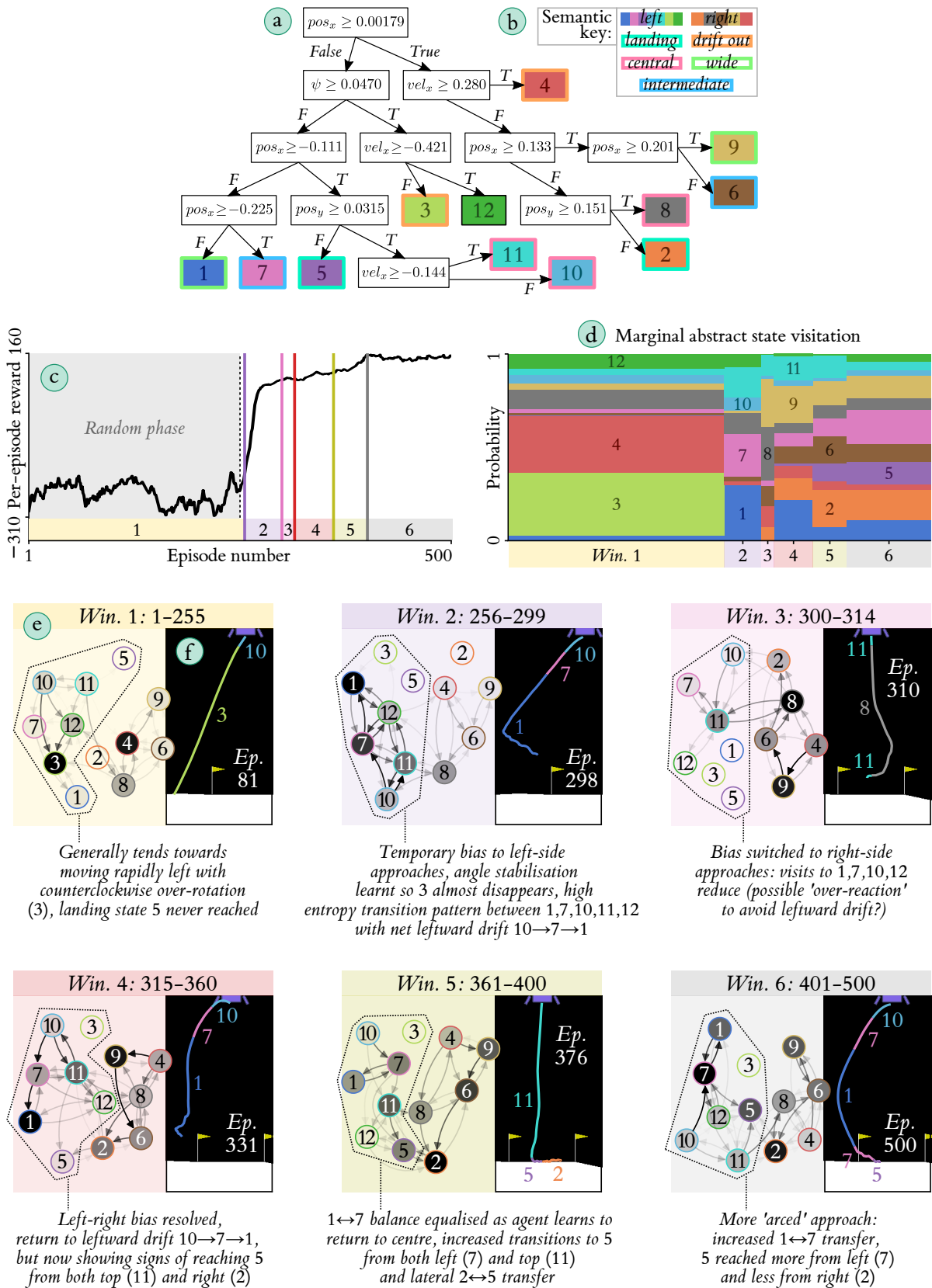


Figure 4.10: Abstraction results and analysis for LunarLander environment.

4.6.4 Window Prototypes

The preceding interpretation of transition graphs is reinforced by visualising a *prototype* episode **f** for each window w . We define this as the one that is most representative of the window’s aggregated dynamics under the log-likelihood of conditional transition probabilities:

$$(4.11) \quad \text{prototype}(w) = \operatorname{argmax}_{l_w \leq i < u_w} \left[\frac{1}{H^i} \sum_{t=0}^{H^i-1} \log P_{w,x_{i,t},x_{i,t+1}}^{\mathcal{X},\mathcal{W}} \right],$$

where H^i is the length of the i th episode, and $x_{i,t}$ is the abstract state containing the state feature vector for the t th timestep of that episode. The prototypes generally agree well with the transition graph annotations for each window, with the one for window 1 providing a clear example of the uncontrolled leftward drift, and those for windows 5 and 6 illustrating the change from directly vertical landing approaches to ‘arced’ ones.

Summarising agent policies using prototypical exemplar trajectories is a popular approach, as embodied by the HIGHLIGHTS [6] and contrastive DISAGREEMENTS [8] frameworks. It is notable that we obtain this mechanism for prototype generation ‘for free’ as a by-product of building an abstract transition model. Also note that the log-likelihood objective is the same as that used in the Dijkstra search method in Section 3.9.3, although here it is being employed to search over actual agent trajectories rather than synthesise hypothetical ones.

4.6.5 Posterior analysis and Counterfactual Review

A similar calculation allows us to incrementally assess the representativeness of an ongoing episode with respect to each window as time progresses, and develop a mechanism for *counterfactual review* of critical transitions. Given a sequence of abstract state transitions $x_t \rightarrow x_{t+1}$ for an episode (which need not be in the dataset used for learning the abstraction), this analysis involves calculating the log posterior over the conditional transition distribution,

$$(4.12) \quad \text{representativeness}(w, t) = \left[\log \rho_w^{\mathcal{W}} + \sum_{t'=0}^{t-1} \log P_{w,x_{t'},x_{t'+1}}^{\mathcal{X},\mathcal{W}} \right] \quad \begin{array}{l} \forall w \in \mathcal{W}, \\ t \in \{1, \dots, H\}, \end{array}$$

where H is the episode length. Plotting these values as a time series indicates how representative the episode is of each window’s dynamics and how this representativeness evolves over time.

To perform a counterfactual review of critical transitions in the episode, we identify alternative abstract states that would have significantly altered the posterior if they had been transitioned to at a particular timestep. This provides a notion of locally relevant contrasts between windows, which are grounded in the events of a particular episode. This local analysis is complementary to the more globally oriented discussion of window contrasts in previous sections, and may be helpful for understanding how an agent’s learning changes how it behaves in specific contexts of interest.

For example, a crash landing episode from window 1 is shown in Figure 4.11 **a**. The time series of posteriors for each window **b** (with the value for the true window 1 subtracted as a

baseline) shows that the $6 \rightarrow 4$ transition at $t = 48$ eliminates the possibility of this episode being from any of windows 3, 4 or 5. Windows 2 and 6 remain in contention until the crash terminates the episode at $t = 66$, but the latter is ruled out by the final $4 \rightarrow \emptyset$ transition (i.e. the agent never terminates in abstract state 4 near the end of learning) This leaves only windows 1 and 2, with the true one 1 being more probable (recall the left-side bias in window 2, annotated in Figure 4.10, which makes this right-side episode less representative). The baselined log posterior for window 3 is maximised at $t = 47$ (just before the $8 \rightarrow 6$ transition). A counterfactual review of this transition gives further insight. The outbound conditional probability plot (c) indicates that if the agent had instead transitioned to states 10 or 11 (both of which are in the left half of the state space), or terminated in state 8, the posterior for the true window 1 would have dropped to zero, because no such transitions occur in that window. Therefore, the presence of the transition from 8 to 6 (c.f. 10, 11 or \emptyset) is the single most important indicator that this episode came from window 1.

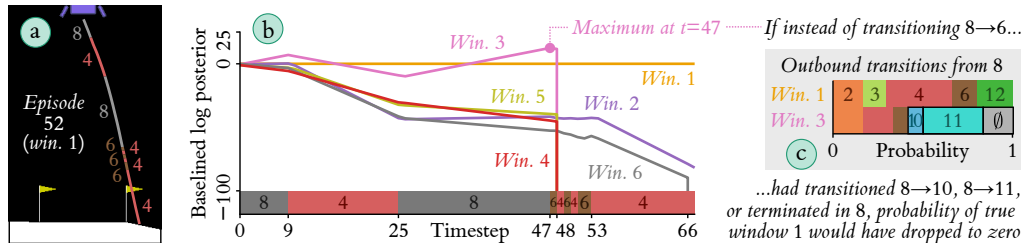


Figure 4.11: Posterior analysis of an unsuccessful (crash) landing during the random phase.

Figure 4.12 (a) depicts a successful landing. The posterior time series (b) shows that transitioning $10 \rightarrow 7$ rules out window 3 as early as $t = 13$. Up to $t = 164$, the most probable window is 2 (likely due to the left-side bias in this period), but this is eliminated by the final transition $12 \rightarrow 5$, leaving the true window 6 as the winner for the remainder of the episode. A counterfactual review (c) indicates that if the agent had instead transitioned to state 11, it would still be representative of window 2 while eliminating window 6. Concretely: a locally-relevant contrast between the two windows is the agent’s ability to touch down on the landing pad (abstract state 5) instead of stay hovering (abstract state 11).

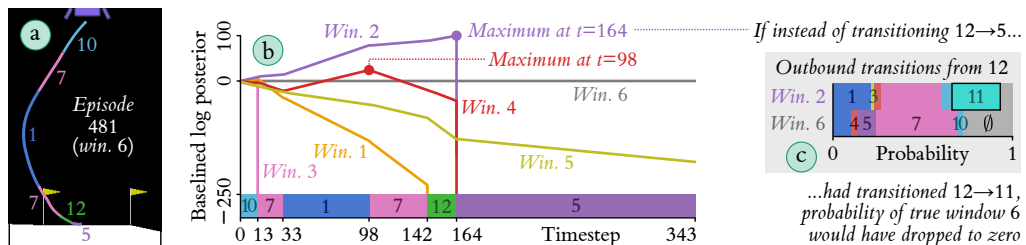


Figure 4.12: Posterior analysis of a successful landing late in learning.

Recent work by Alshehri et al. [4] describes a near-identical technique for incrementally assessing the representativeness of a trajectory with respect to the optimal policies for different reward functions. To our knowledge, our own proposal was developed and published earlier.

4.7 Comparison of Abstraction Algorithm Variants

The preceding abstraction results were produced by the two-stage CSTA algorithm presented in Section 4.3. While this approach is straightforward to describe, implement and debug, it is reasonable to suspect that we could improve both performance on the contrastive abstraction objective (Equation 4.6), and computational efficiency, through well-targeted modifications. For this reason, we now conduct an extensive performance comparison of multiple variants of the original algorithm. The first axis of variation determines the sequence with which the two abstractions \mathcal{X} and \mathcal{W} are grown, as well as the possible introduction of pruning:

- **Grow sequential:** The original sequencing, i.e. first grow the state abstraction \mathcal{X} with fixed windows $\mathcal{W}_{\text{init}}$ then grow the temporal abstraction \mathcal{W} while holding \mathcal{X} fixed.
- **Grow both:** Interleave the growth processes of \mathcal{X} and \mathcal{W} by evaluating possible splits of both abstract states and time windows at every step. $\mathcal{W}_{\text{init}}$ is now only required for the very first step of the algorithm; the first temporal split is reselected from scratch as soon as the first state split has been made.
- **Grow both then prune:** Interleave \mathcal{X} and \mathcal{W} growth as above, but instead of using α and β regularisation to determine stopping, always grow to fixed maximum sizes. ($|\mathcal{X}| = 50$ and $|\mathcal{W}| = 50$ are used below). Then move to a pruning stage, which greedily removes splits from both \mathcal{X} and \mathcal{W} to increase regularised JSD, stopping when no single operation produces an increase. This makes the algorithm similar to CART (Section 3.2.4).
- **Simultaneous:** Further increase the degree of interleaving by removing the distinction between growth and pruning stages. At every step, evaluate all possible splits *and* all possible pruning operations of both \mathcal{X} and \mathcal{W} , and select the one that maximises regularised JSD. This makes it possible for the two abstractions to iteratively grow and shrink in a non-monotonic manner until a local maximum is reached.

We also vary some secondary parameters of the algorithm:

- **Exhaustiveness:** During growth, whether to consider splitting all extant abstract states and/or windows, or only the one containing the most observations from the dataset. (Rationale: non-exhaustiveness greatly reduces runtime, especially for larger abstractions. If performance is similar, this efficiency gain could make the change worthwhile.)
- **Choice of $\mathcal{W}_{\text{init}}$:** As well as $\mathcal{W}_{\text{init}} = \mathcal{W}_{\text{null}}$, we also consider the opposite extreme of just two initial windows, partitioned at the median episode number.
- **Pair-only pruning:** During pruning, whether to evaluate all possible pruning operations (as in M CCP) or only those that remove two ‘sibling’ abstract states/windows at a time.

Excluding invalid combinations, the product of these variables yields a total of 24 variants of the CSTA algorithm, including the original one. We deploy these on three datasets of learning agent transitions: the 2D Maze and LunarLander datasets considered above, as well one for a D4PG RL agent on the benchmark CartPole pole-balancing task, obtained from the

RL Unplugged repository [103]. For each dataset, we explore how the variants perform under different regularisation conditions by varying both α and β within $\{1e^{-4}, 2e^{-4}, 5e^{-4}, 1e^{-3}, 2e^{-3}\}$ (for the 2D Maze dataset) or $\{2e^{-4}, 5e^{-4}, 1e^{-3}, 2e^{-3}, 5e^{-3}\}$ (for the other two datasets). This yields 3 (datasets) \times 5 (α values) \times 5 (β values) = 75 evaluation cases, and for each of these we rank the 24 algorithm variants from best to worst by the regularised JSD of their resultant abstractions. Evaluating in terms of ranks enables a robust like-for-like comparison across all regularisation conditions, between which the absolute magnitudes of JSD values vary. The results are summarised in box plots in Figure 4.13.

The original variant, labelled as **a**, is reliably outperformed by several alternatives. Although its rank tends to be higher on CartPole than the other two datasets, there are still 10 variants that produce stronger results on average. This confirms our suspicion that a two-stage growth strategy, excluding pruning, is not the best possible algorithm for contrastive spatiotemporal abstraction. From the ranking of the 24 variants, we can discern several trends. For example, all else being equal, exhaustive split evaluation during growth is always beneficial. While the non-exhaustive approach is a potentially attractive shortcut from a runtime perspective, these results indicate that this may not be worth the performance impact. Adding a pruning capability is usually beneficial: in all but one case, variants using the ‘simultaneous’ strategy outperform the equivalent variants using the ‘grow both’ strategy. Simultaneous growth and pruning is also usually better than the two-stage ‘grow both then prune’, and for the two exceptions, the overall ranking difference is small. For the higher performing variants, using $\mathcal{W}_{\text{init}} = \mathcal{W}_{\text{null}}$ outperforms starting with a single temporal split at the median episode. However, the trend becomes muddier (or even reverses) for the less successful variants. Another more complex trend is that pair-only pruning makes no difference for the ‘simultaneous’ strategy, while for ‘grow both then prune’, it appears highly beneficial.

The overall best-performing variant, labelled as **b**, consists of simultaneous (exhaustive) growth and pruning, starting with $\mathcal{W}_{\text{init}} = \mathcal{W}_{\text{null}}$. It achieves the highest rank in 27/75 = 36% of evaluation cases and a top-six rank in 57/75 = 76% of them. In Figure 4.14, we compare this ‘best’ variant to the original one across various metrics on all evaluation cases. Column **a** reports the normalised difference in the primary metric of regularised JSD. The ‘best’ variant outperforms the original on all but 5/75 = 6.7% of cases (orange borders), all of which are on the CartPole dataset. The degree of improvement varies markedly between datasets (it is highest for 2D Maze) and across regularisation conditions. Specifically, it appears that the improvement is higher for larger values of α and β , which is to say that the ‘best’ variant offers a greater performance advantage when there is a stronger regularisation towards more compact spatial and temporal abstractions.

Columns **b**, **c** and **d** break the regularised JSD metric into its three components of JSD, abstract state count m and window count n . Importantly, these plots reveal that (except for some LunarLander cases), the improvement in overall performance is *not* due to the ‘best’

4.7. COMPARISON OF ABSTRACTION ALGORITHM VARIANTS

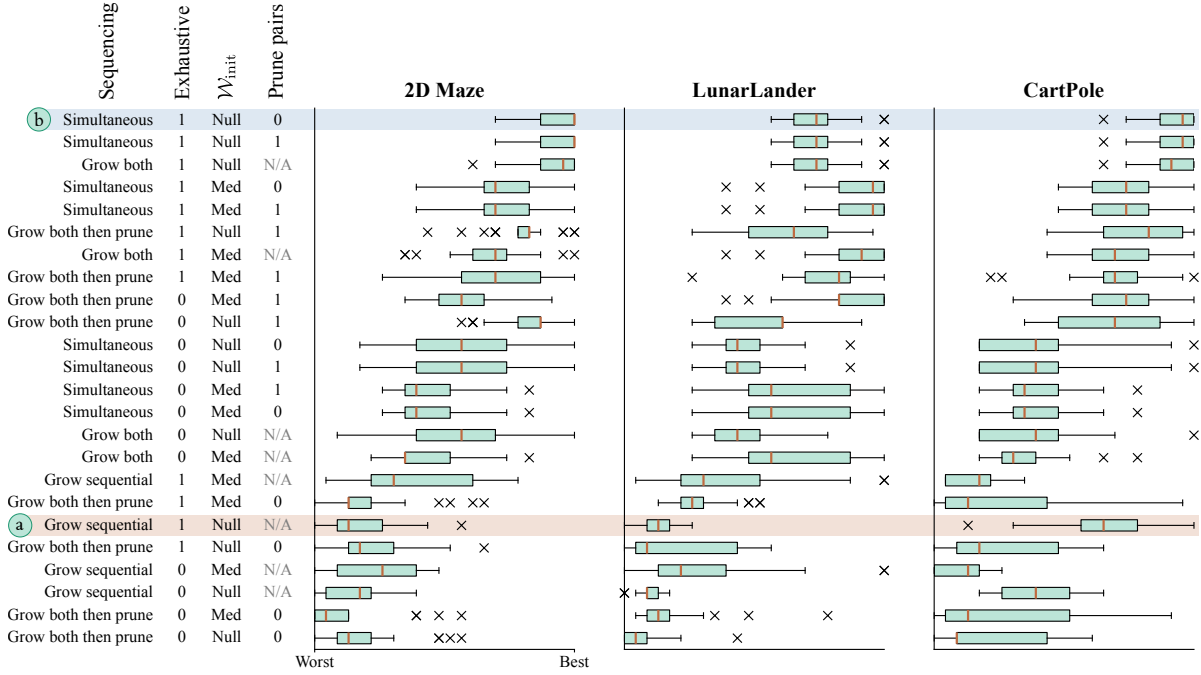


Figure 4.13: Distribution of regularised JSD rankings of 24 algorithm variants across 25 regularisation settings for each of three datasets of learning agent transitions. Variants sorted by median rank across all datasets and regularisation settings (ties broken by mean rank). Box plot whiskers located at $1.5 \times \text{IQR}$ below/above lower/upper quartiles respectively.

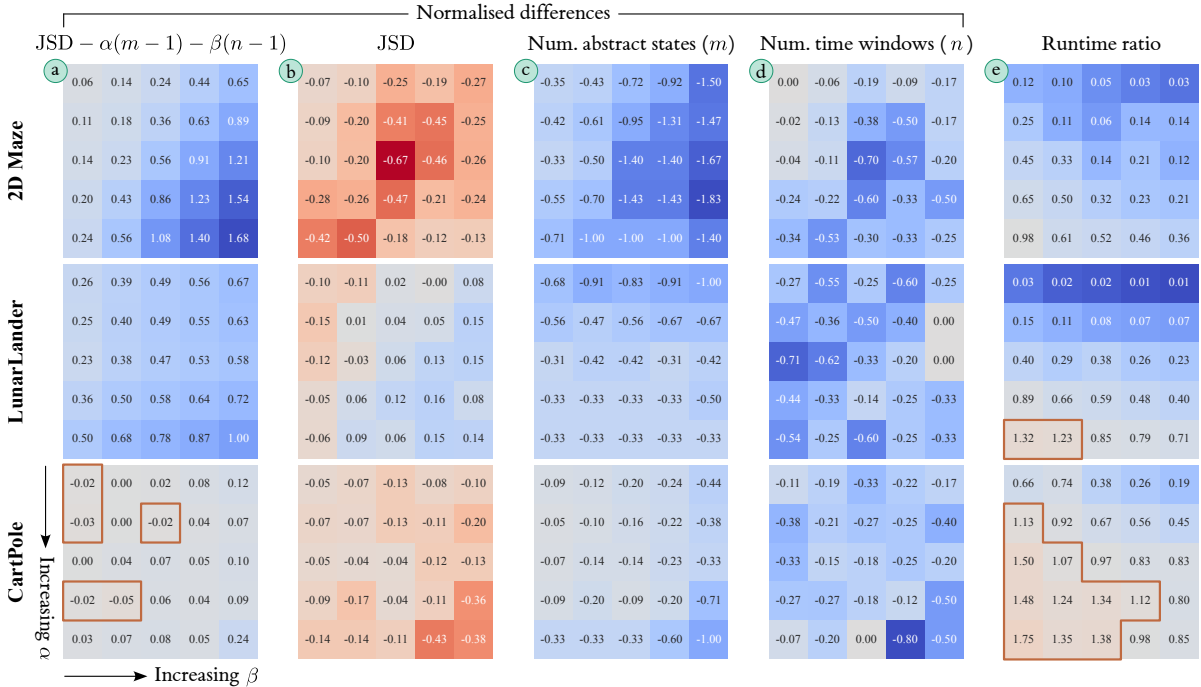


Figure 4.14: Comparison of original and ‘best’ variants across all 75 evaluation cases. First four columns show normalised differences (i.e. (best – original)/best) in the stated metrics, with > 0 indicating that ‘best’ improves (regularised) JSD and < 0 being preferable for m and n . Final column compares runtimes of the two variants; < 1 indicates that ‘best’ is faster.

variant finding abstractions with higher JSD, but rather due to these abstractions being smaller (in both m and n) than those found by the original variant. In fact, across all 75 evaluation cases, the ‘best’ variant *always* yields smaller abstractions for the same regularisation values. In explaining this result, it is difficult to disentangle the combination of changes between the two algorithm variants, but the addition of a pruning capability is likely to be significant.

Finally, column **e** compares the wall-clock runtime (on an Apple M2 MacBook Air) of the two variants on all evaluation cases. The ‘best’ variant is faster in $63/75 = 84\%$ of them. The magnitudes are also significant here: the ‘best’ variant is $100\times$ faster in the best case but only $1.75\times$ slower in the worst case. The most extreme differential occurs when the state abstraction regularisation α is low. In these cases, the first stage of the original algorithm is required to grow the state abstraction to a large size m , all while keeping the maximal set of windows $\mathcal{W}_{\text{init}} = \mathcal{W}_{\text{null}}$. Since the size of the transition probability array is linear in the number of windows n , this array becomes very large towards the end of the growth stage, which in turn makes the calculation of JSD values very expensive. By growing the two abstractions simultaneously and providing a facility for pruning, the maximum size of data structure that must be held in memory, and used in calculations, is reduced.

In the aggregate, these findings support the general superiority of the ‘best’ algorithm variant over the original one. Furthermore, we suggest that moving to an interleaved strategy of simultaneous growth and pruning of both \mathcal{X} and \mathcal{W} makes the algorithm more amenable to online learning from streaming data, because the single stage can be run continually without termination. Implementing and evaluating such an online algorithm would be a valuable direction for future work. Further refinement of the algorithm could yield additional improvements, not least because all variants considered here retain the greedy growth paradigm, without any lookahead to the long-term effect of adding or removing splits.

4.8 Conclusion

In this chapter, we have presented a theory of contrastive spatiotemporal abstraction, which aggregates transition data to generate an interpretable summary of the change points within an agent’s learning process, and developed practical tree-based algorithms for learning such a model. Through experiments with data generated by RL agents in two continuous control environments, we showed how the model reveals the nature and timing of salient dynamic events, such as an agent becoming stuck at, then learning to avoid, an obstacle, or developing and mitigating asymmetries in a landing policy. This information could be conveyed through visualisations such as heatmaps and transition graphs, and via prototype trajectories suggested by the model’s internal statistics. We then examined several variants of the core learning algorithm, leading to an improved proposal that is both faster and more performant on the contrastive objective. This change-oriented approach taken in this chapter offers complementary

insight to the fixed-policy methods in the previous chapter, and may be especially useful for AI practitioners interested in understanding, debugging and stabilising proposed agent learning algorithms. The models and experiments in this chapter have several limitations, which create opportunities for further work:

- Our experiments worked with fixed datasets covering an agent’s learning process from start to end. However, practitioners may be interested in real-time monitoring of changes as they occur, via spatial and temporal abstractions that adapt as new transition data arrive. As noted above, we suspect that adopting a simultaneous growth and pruning strategy would make our algorithms more amenable to learning from streaming data.
- Aside from the choice of $\mathcal{W}_{\text{init}}$, which is later discarded, abstract states and windows are learnt from scratch to maximise the contrastive objective. It may sometimes be desirable to allow a user to ‘seed’ the abstraction with state regions of interest, such as goals and obstacles, or time thresholds when important changes are known to have occurred.
- Somewhat related to the above, information about an agent’s actions or rewards could be stored and used to guide the abstraction process, creating a multiattribute model similar to TRIPLETREE. It would be important to evaluate whether the result is more or less interpretable than a simpler model based on state information only.
- The theory in Section 4.2 is not predicated on using axis-aligned tree models, so there is scope to develop algorithms that learn abstract states with other geometries. This may improve performance on the contrastive objective, although we maintain that there are strong interpretability arguments for the current model architecture (see Chapter 2).
- In the DISAGREEMENTS paper [8], the authors evaluate how their method helps human users to answer questions about qualitative policy differences. It would be valuable to perform a similar user evaluation of our method, baselining against DISAGREEMENTS.

Finally, it is worth reiterating the extreme context-agnosticism of the contrastive abstraction approach. While it was conceived and evaluated in the context of agents that learn over time, it is equally applicable to sequences of policies ordered in any other manner, such as by a performance measure or agent hyperparameter, or (if the temporal abstraction stage is omitted) not ordered at all. The latter case would enable the contrastive analysis of policies that optimise for different objectives, or that are known to originate from different sources such as ‘cautious’ and ‘risk-taking’ agents. Furthermore, because the model does not require action information, it can be applied outside of the agent context to any sequence of Markovian stochastic processes with a common state space, whose transitions can be externally measured. For example, it would be interesting to try using contrastive abstraction to understand changes in time series of weather patterns, medical biomarkers or activity in financial markets.

Chapter 5

Tree Models of Human Preferences

Based on: “Interpretable Preference-based Reinforcement Learning with Tree-Structured Reward Functions”, published at 2022 International Conference on Autonomous Agents and Multiagent Systems.

5.1 Introduction

In previous chapters, we have used tree-based abstraction as a post hoc tool for deriving human-interpretable summaries of an agent’s behaviour and learning. For the following two chapters, we shift perspective to use tree models as a medium through which a human can influence the direction of agent learning itself. Although a human could be brought into the agent learning loop in various ways, we cast our method within the popular formalism of *reward learning* from evaluative feedback [44, 155, 191]. Specifically, we query the human for their preferences over possible agent trajectories for a given task of interest. Then, we learn a tree that predicts preferences for unseen trajectories, which works by assigning scalar reward values to hyperrectangular subsets of a state-action feature space. This tree is then used as a reward function for training the agent to solve the task via RL. Through an iterative process of reward-seeking trajectory generation, preference collection and tree refinement, we aim to converge to both a reward tree that accurately reflects the human’s preferences and an agent policy that satisfies those preferences. Crucially, we also obtain a mechanism for describing and explaining the agent’s learning and actions, which are grounded in the obtained preferences via the tree’s rule structure. Framed this way, a reward tree is no longer just a tool for post hoc interpretation, but a medium through which a human and an agent can understand each other.

In this chapter, we describe our reward tree learning approach. We then quantitatively evaluate it on four benchmark control tasks using real and synthetic human feedback in both offline and online learning settings. Effective and sample-efficient learning of reward trees is observed in each of these contexts, alongside some informative failure cases. We then show how the interpretability properties can be leveraged to explore and debug the learnt trees and the resultant agent policies. First, however, we take a step back to motivate the general strategy of reward learning, and the importance of interpretability in this context.

5.1.1 Motivation

For an RL agent to reliably achieve a goal or desired behaviour for its human designers, owners, operators or customers, its reward function must give a robust numerical representation of that objective. Such a reward function can be extremely challenging to find because human goals and preferences are often tacit, uncertain, hard to formalise or even inconsistent (both within and between individuals). Any mistakes in the reward function are liable to cause undesirable and unsafe agent behaviour. This is known as the reward alignment problem. As RL algorithms are deployed in increasingly complex domains, the design of aligned reward functions may become *“both more important and more difficult”* [70]. The prevailing reliance on manual trial-and-error reward design, which already presents challenges to real-world deployment [155] and hampers the use of RL by non-experts [265], is likely to prove inadequate in the face of vastly more capable agents which can exploit any specification errors to produce highly unexpected outcomes [29, 195]. In a recent survey of manual reward designs for autonomous driving, almost all failed basic ‘sanity checks’ by having missing or redundant features, or failing to distinguish the true objective from heuristic shaping reward for guiding learning [143].

One way to move beyond manual reward design is to reframe RL as an inherently human-in-the-loop problem [244], in which the agent learns its reward function by interacting with a human rather than assuming a fixed objective beforehand [217]. In this vein, methods have been developed for learning rewards from preference feedback provided by the human [155], who is assumed to have a tacit understanding of the desired agent behaviour, and to be capable of recognising a high-quality example when they see it, but is unable to give a precise formal specification. The preference feedback may consist of demonstrations of correct actions [189], or scalar evaluations [146], approval labels [101], corrections [17] or rankings [44] of candidate behaviours proposed by the agent. Reward learning methods show technical promise and have gained visibility due to their efficacy for fine-tuning language models such as INSTRUCTGPT [191] and its successors. However, an oft-unquestioned aspect of the approach creates a roadblock to more widespread application: reward learning typically uses complex, black box model architectures such as neural networks [44], which resist human scrutiny and interpretation, even by those on whose preferences the models are based.

For advocates of interpretability, this is a problematic state of affairs. Performing RL with a learnt reward function yields a policy whose performance depends not only on the expected reward under that reward function, but also on the alignment of the reward to the human’s true preferences. Without explicit knowledge of this ground truth, it is hard to define quantitative metrics for the latter, which instead becomes a subjective, multi-faceted judgement, requiring the human to develop an intuitive understanding of the reward function and its effect on agent learning. If that reward function has been learnt by an uninterpretable black box model, this becomes very challenging. Concerns have been raised about the safety and accountability risks of opaque learning algorithms [214], but an inability to even interpret the objective that an

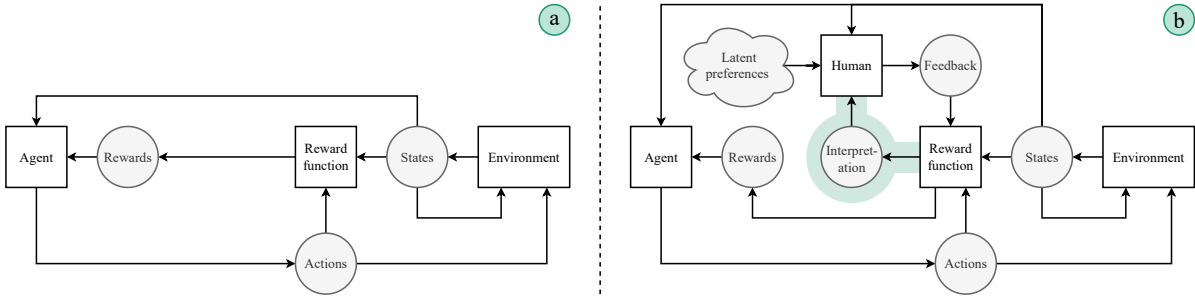


Figure 5.1: In the typical RL formulation (a), the reward function represents a manual attempt to codify a human objective. As such, it is liable to suffer from misalignment. In reward learning (b), the human is brought into the learning loop to provide feedback on the agent’s behaviour. Prior approaches use this feedback to learn a reward function with a black box neural network architecture. By instead using an interpretable tree architecture, we enable the human to intuitively verify the reward learning process and potentially improve their future feedback.

agent is optimising makes it yet harder to understand the causal origins of learnt behaviour and its alignment with human preferences. Black box reward learning could also be seen as a missed scientific opportunity. A learnt reward function is a tantalising object of study for interpretability research because it is simultaneously (1) an *explanatory* model of revealed human preferences, (2) a *normative* model of agent behaviour, and (3) a *causal* link between the two. Finally, for the human engaged in the reward learning process itself, having the ability to interpret a reward model as it is being learnt may help them to provide more targeted and informative feedback in future, thereby establishing a virtuous cycle to improve alignment.

For these reasons, there would be significant value in a method that could learn intrinsically interpretable reward functions from human preference feedback. This chapter proposes and evaluates such a method, which makes use of the language of tree abstractions developed throughout this thesis. Figure 5.1 compares this proposal for interpretable reward learning to the typical RL formulation using a fixed, manually specified reward function.

5.1.2 Related Work

The growing popularity of reward learning has been accompanied by a convergence of model architectures towards powerful but uninterpretable neural networks, which are used in both seminal works [44] and state-of-the-art baselines [154]. However, the importance of interpretability in human-in-the-loop RL has been emphasised in surveys [13, 155], and reward learning and reward explanation have been framed as equally vital components of an interactive alignment process [84]. This fits within the broader framework of *explanatory interactive learning*, which to date has been explored primarily in the supervised learning context [245].

Some early efforts have been made to improve human understanding of reward functions and their effects on action selection. While this area is “*less developed*” than other sub-fields of agent interpretability [98], a distinction has emerged between intrinsic and post hoc approaches.

Proposals for intrinsically interpretable reward functions include those that decompose into semantic components [134], explicitly depend on human-taught features [28], combine a simple linear model with an easily visualised ‘criticality’ weighting [273], or optimise for sparsity by giving zero reward in most states [69]. Post hoc approaches include the application of feature importance analysis [216], counterfactual probing [174], or simplifying transformations [130] to existing reward functions learnt by black box architectures such as neural networks. In [169], a human is asked to provide feedback on the appropriateness of a learnt reward function’s feature importance values, which in turn is used to refine the reward function itself. To our knowledge, this latter work is the closest existing integration of reward learning and explanation. In the intrinsic/post hoc taxonomy, our proposal of reward tree learning is an intrinsic approach, as the rule-based tree structure is inherently readable by humans.

Tree models have found various uses in human-in-the-loop agent learning pipelines, including to learn environment abstractions from human demonstrations [49] and model expert judgements for naval air defence [151]. In [242], a differentiable tree is distilled from natural language to serve as a warm-start RL policy, and in [212], a tree policy is extracted from a trained RL agent before being manually edited to improve performance (note how these two strategies mirror one another). While evolutionary tree algorithms have been used to learn dense intrinsic rewards from sparse environment ones [228], our proposal is the first to learn and use reward trees without any ground truth reward signal, and the first to do so from human feedback.

5.2 Preference-based Reward Learning

Various human feedback signals have been used for reward learning [17, 101, 131, 146, 189]. We adopt the pairwise preference-based approach [44], in which the human is presented with pairs of agent trajectories (sequences of state, action, and next state *transitions*) and identifies which they prefer as a solution to a given task of interest. A reward function is learnt to reconstruct the pattern of preferences by assigning high rewards to transitions in commonly-preferred trajectories and low rewards to those in less-preferred trajectories. This approach is popular [37, 154, 220, 266] and has a firm psychological basis. Experimental results indicate that humans find it easier to make relative (c.f. absolute) quality judgements [138, 264] and exhibit lower variance when doing so [106]. This may be because it avoids the need to maintain an absolute scale in working memory, which is liable to induce bias as it shifts over time [82]. Providing preferences also incurs less cognitive burden than manual demonstrations of desired behaviour [126] and may enable more fine-grained distinctions [27].

To formalise preference-based reward learning, we assume that the human observes a trajectory ξ^i as a sequence $(\mathbf{f}_1^i, \dots, \mathbf{f}_{H^i}^i)$, where $\mathbf{f}_t^i = \phi(s_{t-1}^i, a_{t-1}^i, s_t^i) \in \mathbb{R}^D$ represents a single transition as a D -dimensional feature vector. We take the feature function ϕ as given and assume that all features are individually interpretable. As discussed throughout this thesis,

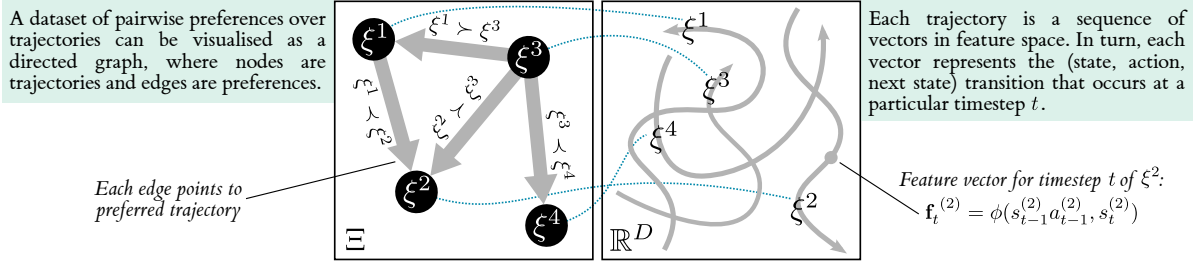


Figure 5.2: Pairwise trajectory preferences as a directed graph.

realistic applications necessitate careful thought about how these features are manually defined or learnt (e.g. via the operator-based feature generation method from Section 3.2.2). Notice that, unlike all prior methods in this thesis, features are now functions of complete (state, action, next state) transitions rather than individual states. This is for the sake of maximum generality; reward functions are defined at the transition level (see Section 1.2), so this formulation ensures that all possible rewards can in principle be represented and learnt.

Assume that there exists a set of N trajectories, $\Xi = \{\xi^i\}_{i=1}^N$, whose provenance is discussed in later sections. Preference-based reward learning begins by consulting the human to obtain $K \leq N(N-1)/2$ pairwise preference labels, $\mathcal{L} = \{(i, j)\}_{k=1}^K$, each of which indicates that the j th trajectory is preferred to the i th as a solution to a given task of interest (denoted by $\xi^j \succ \xi^i$). Figure 5.2 shows how the preference dataset (Ξ, \mathcal{L}) can be understood as a directed graph.

To learn a reward function from this dataset, we must assume a generative model for the preference labels. With a few notable exceptions (e.g. [80]), the existing literature is very consistent on this point [44, 154, 155], and uses the following set of assumptions:

- The human produces labels according to a latent ground truth reward R , which represents their tacit understanding of the task they want the agent to complete.
- The reward function is defined over the single-transition feature space and produces real scalar outputs, i.e. $R : \mathbb{R}^D \rightarrow \mathbb{R}$.
- The perceived quality of trajectory ξ^i is determined by its cumulative ground truth reward (or *return*) $G(\xi^i | R) = \sum_{t=1}^{H^i} R(\mathbf{f}_t^i)$.
- Given two trajectories ξ^i and ξ^j , the human tends to prefer the one with higher return.
- However, the human is liable to make occasional mistakes in their judgement. Mistakes are more likely for trajectory pairs whose returns are more similar.

These assumptions create the following model for the probability of a preference $\xi^j \succ \xi^i$:

$$(5.1) \quad \Pr(\xi^j \succ \xi^i | R) = f(G(\xi^j | R) - G(\xi^i | R)),$$

where $f : \mathbb{R} \rightarrow [0, 1]$ is a non-decreasing *link* function with $f(z) + f(-z) = 1$. The link function encodes how the probability of a mistake is assumed to vary with the difference in return.

Given a preference dataset (Ξ, \mathcal{L}) , the generative model in Equation 5.1, and a choice of link function, the objective of reward learning is to approximate the latent reward function R within some learnable function class \mathcal{R} . This can be formulated as a problem of loss minimisation over predictions of the preferences \mathcal{L} :

$$(5.2) \quad \operatorname{argmin}_{R \in \mathcal{R}} \left[\sum_{(i,j) \in \mathcal{L}} \ell(\Pr(\xi^j \succ \xi^i | R)) \right],$$

where $\ell : [0, 1] \rightarrow \mathbb{R}_{\geq 0}$ is a loss function.

Many methodological variants result from different choices of link function f and loss function ℓ . The popular Bradley-Terry preference model [30] uses the logistic function as f , which is mathematically and computationally simple, but another valid option is the normal cumulative distribution function, which is favoured by Thurstone’s Case V model [246]. For ℓ , the negative log-likelihood loss is most common [44, 154], but Wirth et al. [266] also use a discrete 0-1 loss which considers only the directions of predicted preferences rather than their strengths.

Having used an appropriate optimisation scheme to find a reward function R that (approximately) minimises the chosen preference loss, an agent can use this function to learn a policy by RL. In practice, reward learning and policy learning can be performed in an iterative online manner, with the agent using its latest policy to generate new trajectories to add to Ξ , the human providing new preferences as those trajectories arrive, and the reward function being updated accordingly. Our approach to online reward learning is described in Section 5.4.

5.3 Interpretable Reward Learning with Trees

The choice of reward function class \mathcal{R} is a critical issue. In early prior work, \mathcal{R} was often the class of linear models $R(\mathbf{f}) = \mathbf{w}^\top \mathbf{f}$ [220], which are easy to interpret but have limited expressiveness, so cannot scale to complex tasks. More recently, it has been common to use deep neural networks [44] (or multi-network ensembles thereof), which are far more powerful while remaining tractably learnable by gradient-based methods. However, the complex, multilayered architecture of a neural network resists human scrutiny and interpretation. This motivates exploring an alternative function class that compromises between these two extremes. As hierarchies of simple, interpretable local rules that nonetheless have the theoretical capacity for universal function approximation [170], the axis-aligned tree models studied in this thesis are well-positioned to provide such a compromise.

In this section, we describe our proposal for a tree-structured reward learning model, which we refer to as a reward tree, and outline the core algorithm followed to learn this model from a dataset of pairwise trajectory preferences. Since the discrete split decisions that parameterise an axis-aligned tree are non-differentiable, no tractable gradient-based method exists for optimising a loss function of the form given in Equation 5.2 end-to-end. For this reason, we approximate the

global loss minimisation problem by a sequence of local ones, following a multi-stage learning method with a proxy objective for each stage.

The four stages outlined below, and summarised visually in Figure 5.3, were developed through an iterative process involving several alternative directions, which are signposted where relevant. This approach is computationally efficient, easy to implement, and yields reward functions that are significantly more robust to small data changes than several alternatives that were tried. However, we make no claim of optimality. In the next chapter, we will identify several changes to this method that improve the performance of reward tree learning when applied to a complex industrially-motivated use case.

5.3.1 Trajectory-Level Return Estimation

This first stage of our method considers the N trajectories in Ξ as atomic units, and uses the set of preference labels to construct a vector of return estimates $\mathbf{g} \in \mathbb{R}^N$, which should be higher for more preferred trajectories. The generic form of the proxy objective for this stage is:

$$(5.3) \quad \operatorname{argmin}_{\mathbf{g} \in \mathbb{R}^N} \left[\sum_{(i,j) \in \mathcal{L}} \ell(\Pr(\xi^j \succ \xi^i | \mathbf{g})) \right], \quad \text{where} \quad \Pr(\xi^j \succ \xi^i | \mathbf{g}) = f(\mathbf{g}_j - \mathbf{g}_i).$$

This is a vanilla preference-based ranking problem of the kind routinely faced in AI, psychology and economics, and admits a standard solution. Our method is based on the least squares proposal of Mosteller [182], for which the loss function is $\ell(p) = (f^{-1}(1 - \varepsilon) - f^{-1}(p))^2$ with $\varepsilon \in (0, 0.5)$. We also define $f(z) = \Phi(z)$, the normal cumulative distribution function, thereby adopting Thurstone’s Case V model [246]. These choices specialise Equation 5.3 as follows:

$$(5.4) \quad \operatorname{argmin}_{\mathbf{g} \in \mathbb{R}^N} \left[\sum_{(i,j) \in \mathcal{L}} (\Phi^{-1}(1 - \varepsilon) - (\mathbf{g}_j - \mathbf{g}_i))^2 \right].$$

This loss function incentivises each preferred trajectory ξ^j to have a return that exceeds that of the non-preferred trajectory ξ^i by a positive amount equal to $\Phi^{-1}(1 - \varepsilon)$. This corresponds to the preference $\xi^j \succ \xi^i$ being predicted with probability $1 - \varepsilon$ under Thurstone’s model (note that ε cannot be 0 because $\Phi^{-1}(1)$ is undefined). Throughout this chapter, we use $\varepsilon = 0.1$, which is tantamount to assuming that the human makes a mistake in their preference labelling 10% of the time. Although this is arbitrary, an equivalent assumption is made in [44].

An exact solution to Equation 5.4 can be found via a matrix method proposed by Morrissey and Gulliksen [104]. This method involves constructing a matrix $A \in \{-1, 0, 1\}^{K \times N}$ as follows. Iterating through the preference labels \mathcal{L} in an arbitrary order, let (i, j) be the $k \in \{1, \dots, K\}$ th element obtained. The k th row of A is defined to have -1 in the i th column, 1 in the j th column, and 0 everywhere else. A can be used to express Equation 5.4 in matrix form:

$$(5.5) \quad \operatorname{argmin}_{\mathbf{g} \in \mathbb{R}^N} \left[\|\Phi^{-1}(1 - \varepsilon)\mathbf{1}_K - A\mathbf{g}\|_2^2 \right],$$

where $\mathbf{1}_K$ is the $K \times 1$ vector of 1s. This is the standard form of a least squares problem, so has the following exact solution via the Moore-Penrose pseudoinverse:

$$(5.6) \quad \mathbf{g} = \Phi^{-1}(1 - \varepsilon)(A^\top A)^{-1}A^\top \mathbf{1}_K.$$

As a toy example of this process, the values of A and \mathbf{g} corresponding to the preferences in Figure 5.3 (a) are given below (note that $1.282 = \Phi^{-1}(1 - \varepsilon)$ for $\varepsilon = 0.1$):

$$A = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \implies \mathbf{g} = 1.282(A^\top A)^{-1}A^\top \mathbf{1}_K = \begin{bmatrix} \mathbf{g}_1 = -0.107 \\ \mathbf{g}_2 = 0.748 \\ \mathbf{g}_3 = -0.961 \\ \mathbf{g}_4 = 0.320 \end{bmatrix}.$$

Observe that ξ^2 obtains the highest return estimate since it has two favourable preferences, while the three unfavourable preferences for ξ^3 result in it receiving the lowest estimate. Also note that the final effect of ε is merely to scale the return estimates by a constant factor while leaving their ordering and relative spacing unchanged. Our choice of $\varepsilon = 0.1$ is thus entirely arbitrary and has no significant downstream effects.

5.3.2 Leaf-level Reward Estimation

The vector \mathbf{g} estimates the returns of all trajectories in Ξ , but the ultimate aim of reward learning is to decompose these into sums of rewards for the constituent transitions, then generalise to make reward predictions for unseen data (e.g. new trajectories executed by an RL agent). The core novelty of our method lies in doing this using a tree model over the feature space \mathbb{R}^D , of the kind explored throughout this thesis.

For didactic purposes, let us initially assume that such a tree exists. We will return to how it is constructed in the next subsection. Recall that in our notation, a tree $\mathcal{X} = \{x_1, x_2, \dots, x_m\}$ is a partition of \mathbb{R}^D into hyperrectangular subsets called leaves, where the partition structure is defined by a hierarchy of axis-aligned hyperplanes. The purpose of this second stage of our method is to associate each leaf $x \in \mathcal{X}$ with a real-valued summary statistic, $\text{reward}(x) \in \mathbb{R}$, which will be returned as the predicted reward for any transition (s, a, s') whose feature vector $\mathbf{f} = \phi(s, a, s')$ is contained in x . This is achieved by identifying the subset of trajectories in Ξ that pass through x (i.e. that include at least one feature vector contained in this leaf) and aggregating the corresponding return estimates from \mathbf{g} into a single number.

Concretely, we define $\text{reward}(x)$ as an expectation over \mathbf{g} , weighted by the proportion of transitions from each trajectory whose feature vectors are contained in x :

$$(5.7) \quad \text{reward}(x) = \hat{H} \mathbb{E}_{(i,t) \in \mathcal{D}_x} \left[\frac{\mathbf{g}_i}{H^i} \right], \text{ where } \mathcal{D}_x = \left\{ (i, t) : \mathbf{f}_t^i \in x, \begin{array}{l} \forall i \in \{1, \dots, N\} \\ \forall t \in \{1, \dots, H^i\} \end{array} \right\},$$

and $\hat{H} = \sum_{i=1}^N \frac{H^i}{N}$ is the mean trajectory length in Ξ . This definition assigns higher reward to leaves that contain more transitions from trajectories with high \mathbf{g}_i values, which in turn are

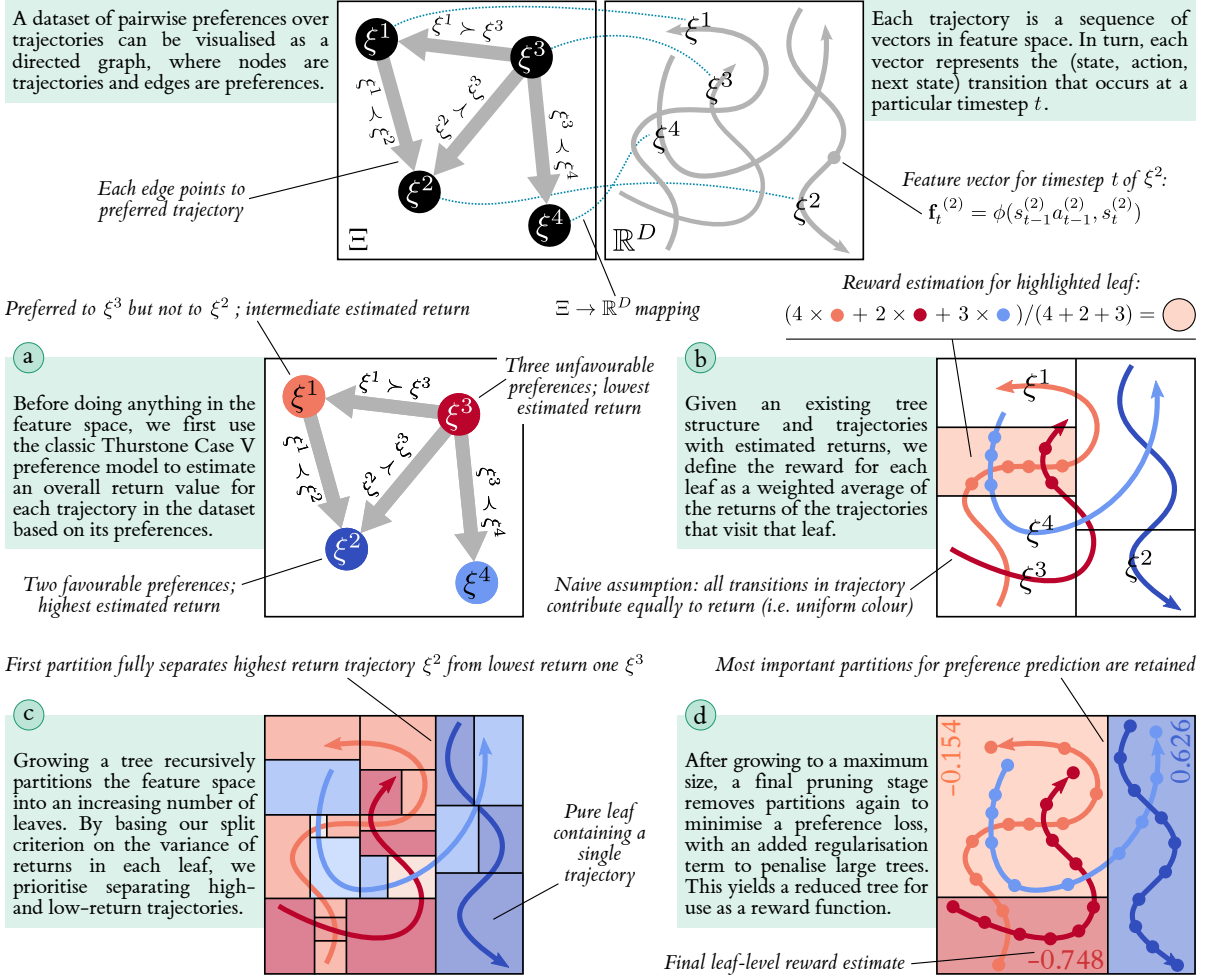


Figure 5.3: The four stages of reward tree learning applied to a toy example of $K = 4$ preferences over $N = 4$ trajectories in a $D = 2$ -feature space (Figure 5.2 duplicated above for reference).

those more commonly preferred in the preference dataset. While this approach of treating every transition as an equal contributor to its trajectory’s return is ostensibly naïve, we find that it enables an effective, robust and tractable reward learning algorithm. It minimises the number of free parameters in subsequent learning stages (thereby permitting fast implementation) and provides an intuitive interpretation of predicted reward that is traceable back to a \mathbf{g}_i value and set of transitions for each $\xi^i \in \Xi$.

To continue the toy example, Figure 5.3 (b) shows how 4, 2 and 3 transitions from ξ^1 , ξ^3 and ξ^4 are averaged over to yield the reward estimate for one of five leaves in a tree, indicated by the orange shading. Assuming that all trajectories contain the same number of transitions so the factor \hat{H}/H^i in Equation 5.7 cancels, the specific numerical calculation is as follows:


$$\text{reward}(x) = \frac{4 \times -0.107 + 2 \times -0.961 + 3 \times 0.320}{4 + 2 + 3} = -0.154.$$

Before converging on this time-weighted averaging approach to estimating leaf rewards, we

assessed the viability of two other methods, which we briefly discuss now:

- **Least squares:** We first tried formulating a second least squares problem (i.e. in addition to the one used for return estimation) to find leaf-level rewards. This involved solving for the $m \times 1$ vector $\mathbf{r} = (C^\top C)^{-1} C^\top \mathbf{g}$, where C is the $N \times m$ matrix of counts of the timesteps each trajectory spends in each leaf. We found that this approach tended to give brittle and poorly regularised results, with extremely high or low reward magnitudes assigned to leaves that were only visited by especially high- or low-return trajectories (even if this was just for a single timestep) and near-zero reward for many others. Time-weighted reward assignment makes such extreme values far less likely.
- **TrueSkill:** We also tried bypassing the estimation of trajectory-level returns \mathbf{g} entirely, and instead deriving leaf-level rewards directly from the preference data. For this, we used the Microsoft TrueSkill algorithm [114], originally developed to assign skill scores to individual video game players based on team-level match results. It is grounded in the equations of Thurstone’s preference model. In our appropriation of TrueSkill, the ‘players’ were the leaves of the tree, ‘skill scores’ were reward values, ‘teams’ were trajectories, ‘match results’ were pairwise preference labels, and the algorithm’s *partial play* feature was harnessed to weight each leaf x ’s contribution to a trajectory ξ^i by the number of timesteps spent in it. The mapping of features of the algorithm into our problem context was thus remarkably direct, and we found that it produced plausible and robust reward estimates for a fixed set of leaves. Crucially, however, the algorithm could not readily be adapted to handle the splitting or pruning of leaves, which are described in Sections 5.3.3 and 5.3.4 and effectively change the number of players in a team. The only way of implementing this functionality would be to run the full algorithm from scratch whenever a change is made, which would induce a prohibitive runtime on the order of seconds for each split threshold considered during tree growth. This unfortunately meant that TrueSkill was impractical for our purposes. Nonetheless, we suggest that TrueSkill could form a valuable component of a future reward learning framework.

5.3.3 Tree Growth

The preceding two stages describe the translation from a trajectory-level preference dataset to leaf-level reward estimates for a fixed tree structure, but do not discuss how that structure arises. The latter issue is crucial because it determines how effective the reward tree will be as a predictor of the preferences. As in all methods in this thesis, reward trees are generated by a combination of growth and pruning operations. Recall that given a tree \mathcal{X} , one step of growth involves splitting a leaf $x \in \mathcal{X}$ with a hyperplane at a threshold $c \in \mathbb{R}$ along the $d \in \{1, \dots, D\}$ th feature, thereby creating two new leaves $x^{(d \geq c)}$ and $x^{(d < c)}$. Recursive splitting creates an increasingly fine partition of \mathbb{R}^D . Figure 5.3  shows an example of a reward tree with 23 leaves. Leaf-level reward estimates are indicated by the shading.

The proxy objective for the growth stage takes the form of a greedy split criterion for selecting the next split decision at each step. In this chapter, we define the quality of a candidate split x, d, c as follows:

$$(5.8) \quad \mathbb{Q}^G(x, d, c) = |\mathcal{D}_{x(d \geq c)}| \cdot \mathbb{I}^G(\mathcal{D}_{x(d \geq c)}) + |\mathcal{D}_{x(d < c)}| \cdot \mathbb{I}^G(\mathcal{D}_{x(d < c)}) - |\mathcal{D}_x| \cdot \mathbb{I}^G(\mathcal{D}_x),$$

$$\text{where } \mathbb{I}^G(\mathcal{D}) = \hat{H}^2 \mathbb{E}_{(i,t) \in \mathcal{D}} \mathbb{E}_{(j,t') \in \mathcal{D}} \left[\left(\frac{\mathbf{g}_i}{H^i} - \frac{\mathbf{g}_j}{H^j} \right)^2 \right].$$

Here, \mathcal{D}_x and \hat{H} are defined as per Equation 5.7, and $\mathcal{D}_{x(d \geq c)}$ and $\mathcal{D}_{x(d < c)}$ denote the transitions belonging to the two new leaves. \mathbb{Q}^G therefore measures the population-weighted reduction in a per-leaf impurity measure, namely the variance in the (trajectory length-normalised) \mathbf{g}_i values of contained transitions. This value is high when the split separates transitions from trajectories with very different normalised return estimates into different leaves. It is maximised when it creates ‘pure’ leaves containing transitions from one trajectory only (or those with identical normalised returns). Therefore, this criterion corresponds to the proxy objective of predicting the return of a trajectory given the leaves that it visits. One of the pure leaves in Figure 5.3 (c) is indicated by an annotation.

In line with prior methods in this thesis, our reward tree growth algorithm identifies the greedily-optimal x, d, c triplet through an exhaustive search over candidate split thresholds:

$$(5.9) \quad \operatorname{argmax}_{x \in \mathcal{X}, 1 \leq d \leq D, c \in \mathcal{C}_d} \left[\mathbb{Q}^G(x, d, c) \right].$$

We define \mathcal{C}_d as all midpoints between adjacent unique feature values occurring in Ξ .

As the reader may have noticed, this variance-based split criterion is functionally equivalent to the one used in classical regression tree learning with CART (Section 3.2.4). Therefore, we can perform reward tree growth using a virtually unmodified, and highly optimised, regression tree implementation. However, it is important to note that the criterion is only indirectly aligned with the overall reward learning objective of preference loss minimisation (Equation 5.2). In the next chapter, we show that switching to a more direct criterion consistently improves reward learning and RL agent performance in a challenging evaluation domain.

The conditions under which reward tree growth is initiated and terminated within the broader reward learning process are described in Section 5.4. For the present moment, it suffices to say that at the very start of the process, the tree is initialised with a single leaf covering the entire feature space, $\mathcal{X} = \{\mathbb{R}^D\}$, and growth is succeeded by a pruning stage once either no impurity reduction can be realised by any single split or a tree size limit $|\mathcal{X}| = M_{\max}$ is reached.

5.3.4 Tree Pruning

Pruning reduces the size of the tree by removing some of its splits. In the context of reward learning, this can be beneficial for both performance ([247] find that limiting model capacity

can improve generalisation in preference-based reward learning) and human comprehension (in the language of [130], pruning is a form of “*processing for interpretability*”).

Before describing the pruning process, it is helpful to explicitly state the reward function $R_{\mathcal{X}} : \mathbb{R}^D \rightarrow \mathbb{R}$ induced by a given reward tree structure \mathcal{X} :

$$(5.10) \quad R_{\mathcal{X}}(\mathbf{f}) = \text{reward}(\text{leaf}(\mathbf{f})), \quad \text{where} \quad \text{leaf}(\mathbf{f}) = x \in \mathcal{X} : \mathbf{f} \in x.$$

The generic preference loss from Equation 5.2 can therefore be reformulated as follows:

$$(5.11) \quad \sum_{(i,j) \in \mathcal{L}} \ell(\Pr(\xi^j \succ \xi^i | R_{\mathcal{X}})) = \sum_{(i,j) \in \mathcal{L}} \ell(f(\sum_{t=1}^{H^j} \text{reward}(\text{leaf}(\mathbf{f}_t^j)) - \sum_{t=1}^{H^i} \text{reward}(\text{leaf}(\mathbf{f}_t^i)))) \\ = \sum_{(i,j) \in \mathcal{L}} \ell(f(\sum_{x \in \mathcal{X}} \text{reward}(x) \times (|\{\mathbf{f}_t^j \in \xi^j : \mathbf{f}_t^j \in x\}| - |\{\mathbf{f}_t^i \in \xi^i : \mathbf{f}_t^i \in x\}|))).$$

Given the monotonicity constraint on f , the predicted probability of each preference in \mathcal{L} therefore increases with the positive difference between the number of transitions from the preferred trajectory ξ^j and non-preferred trajectory ξ^i in leaves with large positive rewards, and vice versa for negative rewards. In the pruning stage, we adopt a proxy objective that is directly aligned with minimising the loss of these predictions.

Recall that given a tree \mathcal{X} , one pruning operation merges two leaves into one by removing the hyperplane that partitions them. Echoing the MCCP process from Section 3.2.4, let \mathbb{X} denote the sequence of nested subtrees induced by pruning the fully-grown tree from the preceding stage recursively back to its root, at each step removing the single hyperplane that greedily minimises the next subtree’s preference loss according to Equation 5.11. After completing this recursive pruning process, we select the $\mathcal{X}^* \in \mathbb{X}$ that minimises the sum of the preference loss and a regularisation term that encourages small trees:

$$(5.12) \quad \mathcal{X}^* = \underset{\mathcal{X}' \in \mathbb{X}}{\text{argmin}} \left[\sum_{(i,j) \in \mathcal{L}} \ell(\Pr(\xi^j \succ \xi^i | R_{\mathcal{X}'})) + \alpha |\mathcal{X}'| \right],$$

where $\alpha \geq 0$ is a regularisation coefficient. In this stage, we use the same loss and link functions as in the return estimation stage, namely $\ell(p) = (f^{-1}(1 - \varepsilon) - f^{-1}(p))^2$ with $\varepsilon = 0.1$, and $f(z) = \Phi(z/\hat{H})$, corresponding to Thurstone’s Case V model. The rescaling by $1/\hat{H}$ is required to bring values onto the scale of a standard normal distribution.

In the example in Figure 5.3 (d), pruning yields a final tree \mathcal{X}^* with three leaves, whose leaf-level reward estimates (via Equation 5.7) are overlaid. The preference loss for this tree can be calculated as follows:

$$\begin{aligned} \Phi^{-1}(\Pr(\xi^2 \succ \xi^1 | R_{\mathcal{X}^*})) &= \frac{1}{10}(-0.748 \times (0-2) + -0.154 \times (0-8) + 0.626 \times (10-0)) = 0.898; \\ \Phi^{-1}(\Pr(\xi^1 \succ \xi^3 | R_{\mathcal{X}^*})) &= \frac{1}{10}(-0.748 \times (2-6) + -0.154 \times (8-4) + 0.626 \times (0-0)) = 0.237; \\ \Phi^{-1}(\Pr(\xi^2 \succ \xi^3 | R_{\mathcal{X}^*})) &= \frac{1}{10}(-0.748 \times (0-6) + -0.154 \times (0-4) + 0.626 \times (10-0)) = 1.136; \\ \Phi^{-1}(\Pr(\xi^4 \succ \xi^3 | R_{\mathcal{X}^*})) &= \frac{1}{10}(-0.748 \times (0-6) + -0.154 \times (6-4) + 0.626 \times (4-0)) = 0.668; \\ \text{loss} &= (1.282 - 0.898)^2 + (1.282 - 0.237)^2 + (1.282 - 1.136)^2 + (1.282 - 0.668)^2 = 1.635. \end{aligned}$$

This compares to a best-case loss of $\|(1.282 \times \mathbf{1}_K) - \mathbf{A}\mathbf{g}\|_2^2 = 0.547$ for a tree whose leaves are all pure (which would have to be large, yielding a high regularisation penalty), and a worst-case of $4(1.282)^2 = 6.569$ for a tree with a single leaf predicting uniform reward everywhere.¹ The model predicts every preference in the dataset with a probability greater than 0.5. $\xi^2 \succ \xi^3$ and $\xi^1 \succ \xi^3$ receive the highest and lowest probabilities of $\Phi(1.136) = 0.872$ and $\Phi(0.237) = 0.594$ respectively; this ordering is appropriate because ξ^2 is also preferred to ξ^1 .

The pruned tree \mathcal{X}^* is returned as the final output of the four-stage process and the associated $R_{\mathcal{X}^*}$ can be used as a reward function for training an agent by RL. To the extent that the model is a reliable predictor of human preferences over unseen trajectories, the agent is incentivised to learn a behavioural policy that is aligned with those preferences.

5.4 Offline and Online Learning Algorithms

Thus far, we have yet to discuss the origins of the trajectories Ξ , the strategy by which trajectory pairs are sampled for human preference labelling, or how reward tree learning is practically integrated with the learning of a policy by RL. This section addresses each of these issues.

Reward tree learning can be instantiated in two data settings: offline and online. In the offline setting, the underlying trajectory set Ξ remains fixed (having been generated by some prior exploration process), but the label set \mathcal{L} expands as the human is consulted for their preferences using an optimistic active sampling scheme. The final reward tree can then be used to train an RL agent after the fact. In the online setting, reward learning and RL are performed concurrently. Both Ξ and \mathcal{L} are assembled from scratch, with the former being progressively augmented with new trajectories generated by the RL agent itself as it learns a policy in real-time. This can be viewed as another layer of active sampling, as it tends to bias Ξ towards trajectories that the agent believes to satisfy the human’s preferences at each point in learning. In both settings, reward tree learning is entirely agnostic to the internal representations and algorithms used by the agent, making it compatible with any RL approach [155]. We leverage this fact in the next chapter, when we switch to an entirely different RL algorithm to that used in the following experiments.

Algorithms 3 and 4 (overleaf) summarise the processes of offline and online reward tree learning. The differences between the two algorithms are highlighted in the latter through ~~red text~~ (for removals) and **green text** (for additions). In addition, Figure 5.4 shows a flow diagram of the online setting, which indicates the various nested loops of operations that occur.

¹To explain these calculations, in a pure tree, every leaf would be assigned a reward equal to the return estimate for the unique trajectory that visits it, with no averaging over trajectories needed. In turn, the preference probabilities predicted by the tree would be identical to those from using the return estimates themselves, i.e. $\Phi(\mathbf{A}\mathbf{g})$. In a tree with a single leaf, there can be no differences in per-leaf transition counts between trajectories, yielding a prediction of $\Phi^{-1}(p) = 0$ for any preference p .

Algorithm 3 Offline reward tree learning using a fixed trajectory set.

Inputs: Trajectory set Ξ , link function f , loss function ℓ , preference budget K_{\max} , optimism weight λ , tree update frequency K_{tree} , tree size limit M_{\max} , regularisation coefficient α
Output: Final reward tree \mathcal{X}^* (and corresponding reward function $R_{\mathcal{X}^*}$)

- 1: Initialise empty preference set $\mathcal{L} = \emptyset$, uniform optimistic returns $\mathcal{G}_\lambda = 1$, single-leaf tree $\mathcal{X} = \{\mathbb{R}^D\}$
- 2: **while** $|\mathcal{L}| < K_{\max}$ **do**
- 3: Generate sampling matrix W^{off} using G_λ and constraints (Equation 5.14)
- 4: Sample trajectory pair (ξ^i, ξ^j) according to W^{off} and present to human
- 5: Obtain preference (i, j) or (j, i) and append to \mathcal{L}
- 6: **if** $|\mathcal{L}| \% K_{\text{tree}} = 0$ **then**
- 7: Use f and ℓ to compute trajectory return estimates \mathbf{g} from \mathcal{L} (Section 5.3.1)
- 8: Grow tree \mathcal{X} using variance-based split criterion until size limit M_{\max} reached (Section 5.3.3)
- 9: Generate pruning sequence and select \mathcal{X}^* to minimise α -regularised ℓ (Section 5.3.4)
- 10: Update optimistic return $G_\lambda(\xi^i | R_{\mathcal{X}^*})$ for each $\xi^i \in \Xi$ (Equation 5.13)
- 11: **end if**
- 12: **end while**

Algorithm 4 Online reward tree learning with trajectories generated by an RL agent.

Inputs: Trajectory set Ξ , link function f , loss function ℓ , preference budget K_{\max} , optimism weight λ , tree update frequency K_{tree} , tree size limit M_{\max} , regularisation coefficient α , RL algorithm RLALGO, trajectory budget N_{\max} , preference batch frequency N_{batch} , number of convergence episodes N_{conv}
Output: Final reward tree \mathcal{X}^* (and corresponding reward function $R_{\mathcal{X}^*}$), final agent policy π

- 1: Initialise empty trajectory set $\Xi = \emptyset$, empty preference set $\mathcal{L} = \emptyset$, uniform optimistic returns $\mathcal{G}_\lambda = 1$, single-leaf tree $\mathcal{X} = \{\mathbb{R}^D\}$, agent policy π , $K_{\text{prev}} = 0$, $K_{\text{batch}} = 0$
- 2: **while** $|\mathcal{L}| < K_{\max}$ **do**
- 3: **if** $|\Xi| \geq 2$ **then**
- 4: Generate sampling matrix W^{on} using G_λ and constraints (Equation 5.16)
- 5: Sample trajectory pair (ξ^i, ξ^j) according to W^{on} and present to human
- 6: Obtain preference (i, j) or (j, i) and append to \mathcal{L}
- 7: **end if**
- 8: **if** $|\mathcal{L}| \geq 1$ **and** $|\mathcal{L}| \% K_{\text{tree}} = 0$ **then**
- 9: Use f and ℓ to compute trajectory return estimates \mathbf{g} from \mathcal{L} (Section 5.3.1)
- 10: Grow tree \mathcal{X} using variance-based split criterion until size limit M_{\max} reached (Section 5.3.3)
- 11: Generate pruning sequence and select \mathcal{X}^* to minimise α -regularised ℓ (Section 5.3.4)
- 12: Update optimistic return $G_\lambda(\xi^i | R_{\mathcal{X}^*})$ for each $\xi^i \in \Xi$ (Equation 5.13)
- 13: **end if**
- 14: **if** $|\mathcal{L}| - K_{\text{prev}} = K_{\text{batch}}$ **then**
- 15: Run RLALGO($\pi, R_{\mathcal{X}^*}$) for N_{batch} episodes, yielding new trajectories Ξ_{batch} and updated π
- 16: Store new trajectories, $\Xi \leftarrow \Xi \cup \Xi_{\text{batch}}$
- 17: Compute optimistic return $G_\lambda(\xi^i | R_{\mathcal{X}^*})$ ($= 1$ if $|\mathcal{L}| < K_{\text{tree}}$) for each $\xi^i \in \Xi_{\text{batch}}$
- 18: Update $K_{\text{prev}} = |\mathcal{L}|$ and K_{batch} according to schedule (Equation 5.15)
- 19: **end if**
- 20: **end while**
- 21: Continue to run RLALGO($\pi, R_{\mathcal{X}^*}$) for N_{conv} episodes, yielding final policy π

Both algorithms initiate the tree with a single leaf before collecting a total of K_{\max} preferences, pausing every K_{tree} steps to update the tree on all preferences collected so far via the stages in Section 5.3. The pruned tree \mathcal{X}^* is then used to make optimistic return predictions for all trajectories, which bias the probability of each pair being sampled for future preference

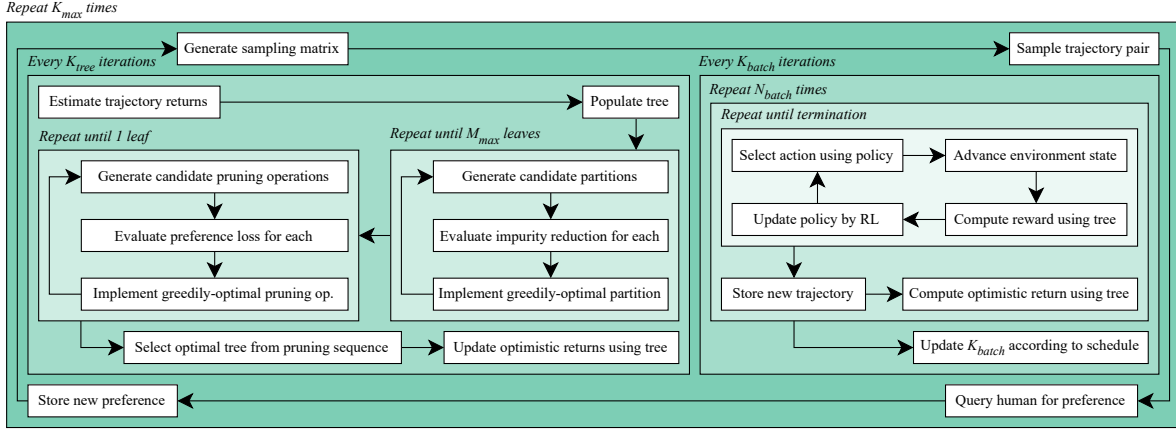


Figure 5.4: Flow diagram of online reward tree learning.

labelling. The critical difference in the online setting is the introduction of an RL agent into the learning loop, which updates its policy π to optimise for the latest $R_{\mathcal{X}^*}$ at each step, and uses that policy to generate trajectories which are appended to Ξ . The rate at which new trajectories are added is scheduled to make the number of preferences obtained for each trajectory more uniform. After K_{\max} preferences have been collected over N_{\max} trajectories, the reward tree is frozen in its final state, and the RL algorithm is run for another N_{conv} episodes to allow the policy to converge. We now clarify some important aspects of the algorithms in further detail.

5.4.1 Trajectory Provenance and Diversity

In the offline context, successful reward learning relies on access to diverse trajectories with both low and high ground truth returns, so their differences can be discovered. One way to achieve this could be to generate Ξ via a reward-free exploration scheme such as DIAYN [83] or ProtoRL [268]. In the online context, the requirement for diversity remains, but the conditions for its achievement are more complex. Ultimately, we require a mix of low and high returns, but the early stages of RL agent training are far more likely to yield the former. The critical condition is whether sufficient early diversity exists for some trajectories to be marginally preferred to others. This establishes a starting point from which the agent can bootstrap to gradually explore and gather preferences on improved behaviours. It is thus vital to use an RL algorithm that promotes exploration early in learning.

5.4.2 Growth Initiation, Stopping and Resumption

Every time K_{tree} new preferences are added in both offline and online settings, the four stages in Section 5.3 are followed to update the trajectory-level return estimates then grow and prune the tree. At the very start of the reward learning process, the tree is initiated with a single leaf and grown from there. In subsequent updates, growth begins from the current state of the tree,

as returned by the previous update, and continues until either there are M_{\max} leaves or none of the candidate splits reduce impurity. This is immediately followed by the pruning stage. At any given point in the process, a subset of the trajectories in Ξ may have received zero preference labels, so have no well-defined return estimate. In practice, each update iteration only makes use of the subset of trajectories with at least one preference, denoted by $\Xi_{\geq 1} = \bigcup_{(i,j) \in \mathcal{L}} \{\xi^i, \xi^j\}$.

5.4.3 Optimistic Active Sampling

Intelligent selection of trajectory pairs for preference labelling can make reward learning more efficient [241]. Inspired by the bandit literature [38], we adopt an upper confidence bound strategy, which samples trajectory pairs according to optimistic estimates of their summed return under the current reward tree. To compute an optimistic return for each trajectory $\xi^i \in \Xi$, we increase each leaf’s reward prediction by the square root of its impurity:

$$(5.13) \quad G_\lambda(\xi^i | R_{\mathcal{X}^*}) = \sum_{t=1}^{H^i} \text{reward}_\lambda(\text{leaf}(\mathbf{f}_t^i)), \quad \text{where } \text{reward}_\lambda(x) = \text{reward}(x) + \lambda \sqrt{I^G(\mathcal{D}_x)},$$

and $\lambda \geq 0$ is an optimism weighting coefficient. This has the effect of up-weighting trajectories that pass through leaves containing other trajectories with a wide spread of return estimates, which in turn implies high uncertainty in the leaf-level rewards. We then populate a strictly lower triangular $N \times N$ weighting matrix W as follows $\forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, i-1\}$:

$$(5.14) \quad W_{i,j}^{\text{off}} = \begin{cases} 0 & \text{if } ((i,j) \in \mathcal{L} \vee (j,i) \in \mathcal{L}) \vee (\Xi_{\geq 1} \neq \emptyset \wedge \{\xi^i, \xi^j\} \cap \Xi_{\geq 1} = \emptyset), \\ G_\lambda(\xi^i | R_{\mathcal{X}^*}) + G_\lambda(\xi^j | R_{\mathcal{X}^*}) + \delta & \text{otherwise,} \end{cases}$$

where the ‘zeroing’ conditions prevent sampling a pair twice and ensure that one of each sampled pair is in $\Xi_{\geq 1}$.² The offset δ is calibrated so that the minimum element not matching a zeroing condition is set to 0.³ In the offline setting, a trajectory pair ξ^i, ξ^j is then sampled for preference labelling by normalising W^{off} by its grand sum and treating the result as a probability distribution. A small modification is made for the online setting (see next subsection).

This strategy prioritises trajectories with uncertain return predictions, for which further preference labels will likely be most helpful in reducing uncertainty. Additionally, the optimism induces a bias towards identifying and correcting cases where trajectory return is overestimated, ultimately yielding a conservative reward function that counteracts the well-known overestimation bias in many RL methods [89]. Finally, biasing the preference dataset towards promising trajectories leads the reward tree to prioritise distinguishing between high and very high return behaviour (rather than low and very low), which reduces the risk of an RL agent trained on that reward stagnating at mediocre performance with no incentive to improve.

²The latter condition ensures that the graph representing the preference dataset is always connected. This is necessary for the least squares solution in Equation 5.6 to be unique; see Proposition 3.1 of [55].

³Unless this is also the maximum element, in which case it is offset to an arbitrary positive value.

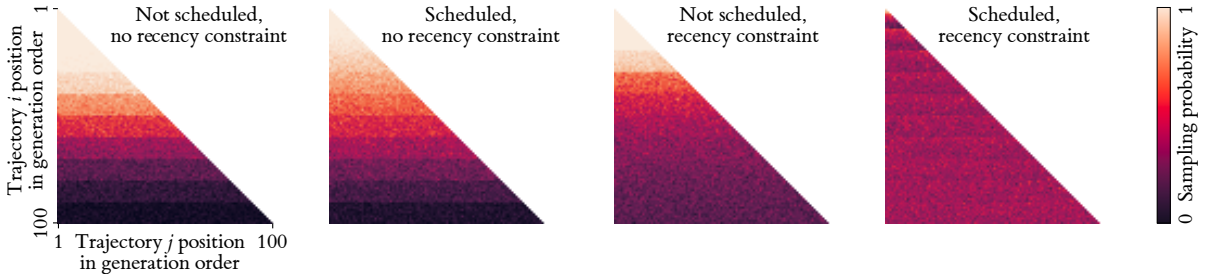


Figure 5.5: Empirical pair sampling probabilities (averaged over 100 repeats) for $K_{\max} = 2000$ preferences over $N_{\max} = 100$ trajectories, which arrive in batches of size $N_{\text{batch}} = 10$. The preference batch size scheduling (Equation 5.15) and the recency constraint (Equation 5.16) must be implemented in tandem to correct the earliness bias fully and achieve uniformity.

5.4.4 Scheduling of Online Trajectory and Preference Collection

In the online setting, Ξ monotonically expands with new trajectories over time. If preferences are obtained at a constant rate, trajectory pairs that appear earlier are more likely to be sampled, creating an earliness bias in the preference dataset.⁴ By expending most of the preference budget on (likely low-quality) trajectories early in the agent’s learning, we forego the opportunity to compare more medium- and high-quality trajectories from later in learning, and thus to learn the finer distinctions required for the agent to find a near-optimal policy. We correct for this bias by collecting an increasing-sized batch of preferences each time N_{batch} new trajectories are added. At each step, the size of the next batch increases monotonically as

$$(5.15) \quad K_{\text{batch}} = \text{round} \left((K_{\max} - |\mathcal{L}|) \frac{\text{pairs}(|\Xi|) - \text{pairs}(|\Xi| - N_{\text{batch}})}{\text{pairs}(N_{\max}) - \text{pairs}(|\Xi| - N_{\text{batch}})} \right),$$

where $\text{pairs}(n) = n(n - 1)$ is the number of pairs that can be sampled from a population of n trajectories. Additionally, the un-normalised sampling probability matrix W^{on} is defined similarly to W^{off} , except with an extra zeroing condition which ensures that at least one of every sampled pair comes from the most recent batch of trajectories:

$$(5.16) \quad W_{i,j}^{\text{on}} = \begin{cases} 0 & \text{if } (i \leq |\Xi| - N_{\text{batch}}) \wedge (j \leq |\Xi| - N_{\text{batch}}), \\ W_{i,j}^{\text{off}} & \text{otherwise,} \end{cases}$$

As Figure 5.5 shows, both modifications are required to ensure uniform sampling probability across all pairs in a population of sequentially generated trajectories. In preliminary experiments, we found that scheduling preference collection in this way improves the performance of converged agent policies across the evaluation domains used in this chapter.

⁴In a constant schedule, K_{\max}/N_{\max} preferences are obtained each time a trajectory is added to Ξ . Let ξ^i and ξ^j be the i th and j th trajectories added. Ignoring the effect of optimistic sampling and the zeroing conditions that prevent duplicates and ensure connectivity, the expected number of times ξ^i, ξ^j is sampled is

$$\sum_{n=\max\{i,j\}}^{N_{\max}} \frac{K_{\max}/N_{\max}}{\text{pairs}(n)} = \frac{K_{\max}}{N_{\max}} \sum_{n=\max\{i,j\}}^{N_{\max}} \frac{1}{n^2 - n} = \frac{K_{\max}}{N_{\max}} \frac{N_{\max} - (\max\{i,j\} - 1)}{N_{\max}(\max\{i,j\} - 1)} \propto \frac{N_{\max}}{\max\{i,j\} - 1} - 1,$$

which decreases as $\max\{i,j\}$ increases. Hence, earlier trajectory pairs obtain a higher density of preferences.

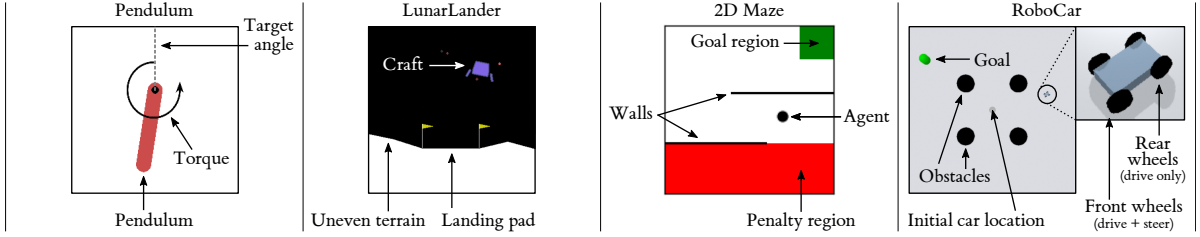


Figure 5.6: The four RL environments used in experiments.

5.5 Experimental Setup

We evaluate reward tree learning in four RL environments under various learning conditions covering both offline and online settings. The results in Section 5.7 use preferences collected from real human experimental participants. First, however, Section 5.6 uses preferences provided by synthetic *oracles*, which act in error-free accordance with hand-coded ground truth reward functions. This combined oracle- and human-based approach allows us to compare how our method performs with and without any possible sources of human error, bias and variability, which inevitably make the learning problem more challenging. This section describes the four environments and their associated tasks, features and ground truth reward functions, as well as common parameters that are held constant across all experiments.

5.5.1 Environments and Tasks

Figure 5.6 contains annotated visualisations of the four environments. All are equipped with natural sets of task-relevant features, which are exposed as part of their native implementation (in Python), and have episodes of a fixed length H , without early termination conditions.

Pendulum A built-in component of the Gymnasium library [86] and a classic feedback control problem. The task is to swing a randomly initialised inverted pendulum upright and hold it there for as long as possible up to a time limit of $H = 200$. The $D = 4$ transition features $\phi(s_{t-1}, a_{t-1}, s_t) = [\cos(\theta_t), \sin(\theta_t), \dot{\theta}_t, a_{t-1}]$ are the cosine and sine of the pendulum’s angle from upright θ_t and its angular velocity $\dot{\theta}_t$, as well as the agent’s latest action $a_{t-1} \in [-1, 1]$, which is a torque on the rotational joint. The ground truth reward function used in Gymnasium is

$$(5.17) \quad R(\phi(s_{t-1}, a_{t-1}, s_t)) = -(\theta_t^2 + 0.1\dot{\theta}_t^2 + 0.001a_{t-1}^2).$$

LunarLander Another built-in Gymnasium environment, as used in earlier chapters. The $D = 18$ transition features consist of all eight features from each of the two successive states, as well as the two action dimensions specifying main and side engine activations: $\phi(s_{t-1}, a_{t-1}, s_t) = [pos_{t-1}^x, pos_{t-1}^y, vel_{t-1}^x, vel_{t-1}^y, \psi_{t-1}, vel_{t-1}^\psi, c_{t-1}^l, c_{t-1}^r, a_{t-1}^m, a_{t-1}^s, pos_t^x, pos_t^y, vel_t^x, vel_t^y, \psi_t, vel_t^\psi, c_t^l, c_t^r]$ (see Section 3.10 to recap state feature definitions). The task is to land the craft gently on a

landing pad. For most timesteps, the ground truth reward function returns

$$(5.18) \quad R(\phi(s_{t-1}, a_{t-1}, s_t)) = \text{potential}_t - \text{potential}_{t-1} - 0.3a_{t-1}^m - 0.03a_{t-1}^s, \text{ where} \\ \text{potential}_t = -100 \left(\sqrt{(\text{pos}_t^x)^2 + (\text{pos}_t^y)^2} + \sqrt{(\text{vel}_t^x)^2 + (\text{vel}_t^y)^2} + |\psi_t| \right) + 10(c_t^l + c_t^r).$$

In addition, a one-off reward of +100 is given if the craft successfully lands on the pad, and -100 is given if it crashes or drifts out-of-bounds ($|\text{pos}_t^x| \geq 1$). To convert LunarLander into a fixed-length episodic task, we disable a default condition that terminates the episode immediately after a landing, crash or out-of-bounds event, and instead use a constant time limit of $H = 300$.

2D Maze A modified, fixed-length version of the continuous navigation environment from Chapter 4, in which the task is to escape or avoid a penalty region and move to a goal region while navigating around a pair of impassable walls. The time limit is $T = 200$. The $D = 4$ transition features $\phi(s_{t-1}, a_{t-1}, s_t) = [\text{pos}_t^x, \text{pos}_t^y, a_t^x, a_t^y]$ are the agent’s bounded position $\text{pos}_t^x, \text{pos}_t^y \in [0, 10]^2$ and latest action $a_{t-1}^x, a_{t-1}^y \in [-0.25, 0.25]^2$, which directly specifies its velocity (prior to clipping at walls and maze boundaries). The ground truth reward function is

$$(5.19) \quad R(\phi(s_{t-1}, a_{t-1}, s_t)) = [\text{pos}_t^x \geq 8 \wedge \text{pos}_t^y \geq 8] - [\text{pos}_t^x \leq 3],$$

where $[\cdot]$ is Iverson bracket notation for the indicator function.

RoboCar A custom environment created using the PyBullet physics simulator [53], in which the task is to drive a four-wheeled car to a green goal object while avoiding four black obstacles. The time limit is $T = 200$. The $D = 11$ transition features $\phi(s_{t-1}, a_{t-1}, s_t) = [\text{pos}_t^x, \text{pos}_t^y, \cos(\theta_t), \sin(\theta_t), \text{vel}_t^x, \text{vel}_t^y, \text{dist}_t, \theta_t, c_t^o, a_{t-1}^{th}, a_{t-1}^{st}]$ are the coordinates of the car’s centroid $\text{pos}_t^x, \text{pos}_t^y$, the cosine and sine of its orientation θ_t , its velocity components $\text{vel}_t^x, \text{vel}_t^y$, the distance and bearing in radians to the goal dist_t, θ_t ($\theta_t = 0$ when facing the goal), a binary indicator of contact with an obstacle c_t^o , and the two dimensions of its latest action. These are a throttle a_{t-1}^{th} demand and steering angle a_{t-1}^{st} , which are applied subject to limits and a simple model of drag and mechanical resistance. The car is initialised at $[\text{pos}_0^x, \text{pos}_0^y, \theta, \text{vel}_0^x, \text{vel}_0^y] = [0, 0, 0, 0, 0]$ and the obstacles are always the same, but the goal location is randomised on each episode. The ground truth reward function is

$$(5.20) \quad R(\phi(s_{t-1}, a_{t-1}, s_t)) = -0.05\text{dist}_t - 0.1c_t^o + [\text{dist}_t \leq 2],$$

where the final term (Iverson notation) adds +1 if the car is within a radius of 2 from the goal.

Note that of the four environments, only LunarLander has a ground truth reward function that depends on the predecessor state of each transition s_{t-1} (to compute potential_{t-1}). To ensure that the ground truth can be reconstructed exactly in principle, we provide features of s_{t-1} to the reward tree learning algorithm, but do not do the same for the other three environments.

5.5.2 Common Parameters

Preference budget Since our offline human experiment involves approximately 60 participants, each of whom provides 10 preference labels per environment, we use a preference budget of $K_{\max} = 600$ for all other experiments. This enables a direct comparison of our algorithm’s performance across environments and experiment types. It also reflects a reasonable demand on human labour (on the order of 1 hour, assuming ≈ 10 labels per minute).

Growth and pruning parameters We use a maximum tree size of $M_{\max} = 100$ to terminate growth and a regularisation parameter of $\alpha = 0.01$ during pruning. The final reward structure is sometimes quite sensitive to the latter, which could easily have been tuned for each environment and experimental context. For the sake of simplicity and to avoid cherry-picking, we identify this single value as one that provides good performance across all four environments.

Additional parameters During optimistic sampling of trajectory pairs for preference labelling, we use $\lambda = 2$. Our least squares loss calculations use $\varepsilon = 0.1$, which is equivalent to assuming a 10% preference error rate.

RL algorithm All RL agents trained in our experiments use the soft actor-critic (SAC) algorithm [107], which uses explicit entropy maximisation to promote exploratory policies (thereby addressing Section 5.4.1). We use a constant discount factor of $\gamma = 0.99$, learning rates of $1e^{-4}$ and $1e^{-3}$ for the policy and value networks respectively, an entropy regularisation coefficient of 0.2, and an interpolation factor of 0.99 for Polyak averaging of the target networks. All networks have two hidden layers of 256 units and ReLU activations. The replay buffer capacity, minibatch size and total number of training episodes were independently selected for each environment after an informal search and held constant across all experiments:

	Pendulum	LunarLander	2D Maze	RoboCar
Replay buffer capacity	5000	20000	20000	40000
Minibatch size	32	64	128	64
Total training episodes (N_{total})	200	200	400	1000

Note that for the online experiments, $N_{\text{total}} = N_{\max} + N_{\text{conv}}$ because agent training consists of a phase of online trajectory/preference gathering and reward tree modification (up to N_{\max}) followed by training to convergence on the frozen final reward tree (another N_{conv} episodes).

Repeated experimental runs In all experiments other than the most labour-intensive online study with human preferences, we repeat five independent runs of RL agent training for each environment and report mean, minimum and maximum performance in the learning curve plots (Figures 5.7-5.10). In the offline setting, all repeats use the same learnt reward tree, so the variation reflects the stochasticity of the RL process only. In the online setting, a reward tree is constructed from scratch using trajectories generated during each agent’s training. Since this process naturally differs between runs, it is an additional source of variation compared with the offline experiments.

5.6 Quantitative Performance: Oracle Preferences

This section presents results from our experiments using synthetic oracle preferences with respect to the ground truth reward functions given in Section 5.5.1. Given a trajectory pair, an oracle always prefers the one with higher ground truth return, making no mistakes and breaking ties uniform-randomly. The use of oracles is widespread in reward learning research [44, 101, 159, 207] as it enables scalable systematic comparison, with the ability to quantify performance (and in our case, appraise learnt tree structures) in terms of the reconstruction of a known ground truth. In particular, we can evaluate an RL agent trained using a learnt reward tree by its return under the ground truth reward function, and baseline this against the return of an agent trained directly on the ground truth itself. The performance gap indicates the degree of misalignment accrued during the preference-based learning process.

Given the highly idealised operation of the oracles, the results in this section are far from a complete guarantee of our method’s efficacy for realistic human-in-the-loop applications. Rather, they serve to establish the basic correctness and stability of reward tree learning from error-free data, before considering real human preferences in the next section.

5.6.1 Offline Setting

We first evaluate oracle-based reward tree learning in the offline setting. For each of the four evaluation tasks, we begin by training an agent via conventional RL on the ground truth reward function. We call this the *pilot* agent. The pilot serves a dual role as (1) a performance baseline for agents trained with learnt reward trees and (2) a generator of the offline trajectory set Ξ . Specifically, we define Ξ as the set of all trajectories executed by the pilot agent during its training. While this use of pilot trajectories would be impractical for realistic applications where the ground truth is unknown, it is a convenient way of ensuring that Ξ contains a diverse spread from low-return trajectories (from early in training) to high-return ones (from late in training). We consult the oracle to obtain $K_{\max} = 600$ preferences over the pilot trajectories, pausing every $K_{\text{tree}} = 60$ steps to update the reward tree and optimistic return estimates G_λ . We then perform five independent RL training runs using the final reward tree.

Figure 5.7 shows learning curves (time series of return per episode) for these RL agents according to both the reward tree itself (a) and the ground truth reward function (b) (mean and min-max range across five repeats shown). The consistent monotonicity and low variance of the former indicate that reward trees give rise to intrinsically stable agent learning on all tasks, while the latter confirm that performance on the extrinsic measure of ground truth return also improves. This shows that the reward trees are at least somewhat aligned. To quantify this, (b) also shows the learning curves of the pilot agents (purple). For Pendulum and RoboCar, the asymptotic returns of agents using the reward trees are indistinguishable from those of the pilots, and training progression even seems to be somewhat more stable and monotonic. Aligned reward tree learning has thus been successfully achieved in these cases. For LunarLander and

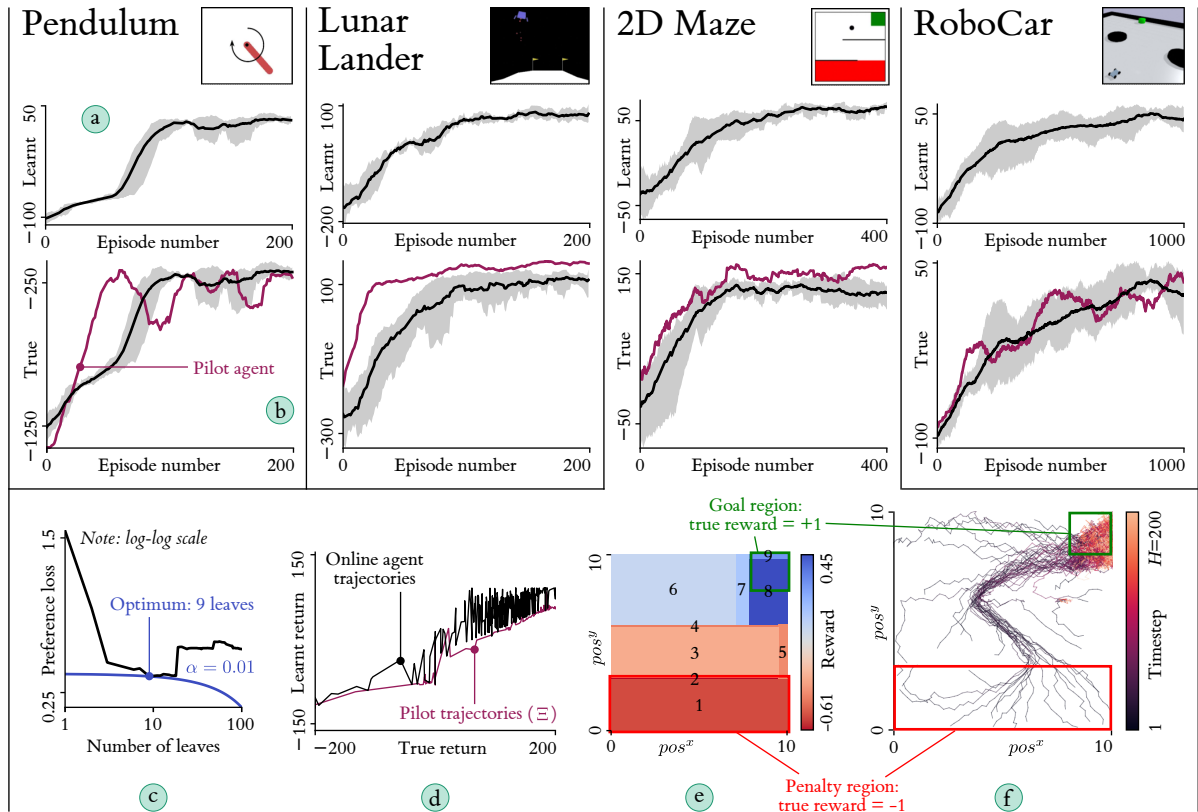


Figure 5.7: Performance in offline setting using oracle preferences; additional plots for 2D Maze.

2D Maze, there is a small performance gap, indicating that alignment is good but imperfect.

Additional plots for 2D Maze provide further insight. (c) plots the preference loss across the tree pruning sequence of the final update step, showing that the (α -regularised) loss is minimised by a tree with 9 leaves. Hence, this is chosen as the final tree. (d) assesses alignment by relating the ground truth and tree-predicted returns of all pilot trajectories Ξ , as well as those generated online during follow-up agent training. In both cases, there is a clear positive correlation, although the relationship for the latter is noisier. In particular, there is a tendency to overpredict the returns of the online trajectories compared with the pilot ones.

To diagnose this issue, (e) uses the projection method from Section 3.10.1 to visualise the geometry of the 9 leaves of the final tree as coloured rectangles over the two positional features pos^x, pos^y , with leaf-level reward as the colouring statistic. In general, the leaves are arranged isomorphically to the maze layout, with negative reward in the penalty region and positive reward around the goal. Several leaf boundaries are aligned almost exactly with the two maze walls. However, one source of misalignment is that high-reward leaf 8 is too large in the vertical direction. (f) plots the final 10 trajectories of all five RL training runs, showing that this misalignment leads to policies that sometimes terminate just below the true goal. We suspect that leaf 8 has not been split further because the pilot dataset contains few examples of trajectories that pass through the below-goal region without then reaching the goal.

5.6.2 Online Setting

We now deploy our algorithm in an online setting, populating Ξ with trajectories generated by an RL agent as it is trained on the continually updated reward tree in real-time. After N_{\max} trajectories and $K_{\max} = 600$ oracle preferences are gathered, the reward tree is frozen, and the agent continues to train on this fixed objective for N_{conv} more episodes to allow its policy to converge. We tune N_{\max} and N_{conv} for each environment as follows:

	Pendulum	LunarLander	2D Maze	RoboCar
N_{\max}	100	100	100	200
N_{conv}	100	100	300	800

In all cases, we use $N_{\text{batch}} = 10$, schedule K_{batch} as per Equation 5.15, and set $K_{\text{tree}} = K_{\text{batch}}$ (i.e. one tree update per batch of trajectories collected). The entire online learning process is repeated five times for each environment.

The learning curves of tree-predicted return in Figure 5.8 (a) indicate that stable and convergent agent training is achieved in all cases, even as the reward trees are modified over time by the discrete operations of growth and pruning. Technically, this makes the RL problem nonstationary, but it appears that the SAC agents can handle this without significant issues. The ground truth learning curves (b) show that asymptotic performance is similar to the oracle-based offline setting (red curves), with the agents’ mean return at the end of learning being slightly higher for 2D Maze, slightly lower for LunarLander and RoboCar, and virtually identical for Pendulum. This suggests that online reward tree learning is no more inherently challenging than learning from diverse offline trajectories. However, learning in the online context tends to exhibit slower performance improvements in the early stages, most notably for Pendulum. This makes sense because the tree may not have yet converged to a well-aligned state, meaning early learning is in a partially misaligned direction. As more preferences are gathered over the early trajectories, this error is gradually corrected.

We further examine one of the five RoboCar runs via a hybrid visualisation that we call a *learning timeline* (c). With $N_{\max} = 200$ and $N_{\text{batch}} = 10$, we run a total of 20 preference-gathering batches, over which the cumulative number of preference labels $|\mathcal{L}|$ increases to 600 according to the K_{batch} schedule. The heatmap shows how the α -regularised preference loss varies as a function of leaf count across the pruning sequence during each post-batch tree update. The overlaid white curve indicates the size of the optimal tree selected from this sequence (i.e. the one that minimises the regularised loss). As the tree is grown and pruned using an increasing number of preferences, the global pattern is that (1) a very low preference loss becomes harder to achieve and (2) the optimal number of leaves starts small before increasing to a maximum, then remains somewhat below that maximum thereafter, with large changes becoming less frequent. We observe this trend consistently across runs and environments.

Pausing to inspect the tree at three checkpoints during learning, we find that the positive correlation between ground truth and tree-predicted return (d) becomes less noisy over time,

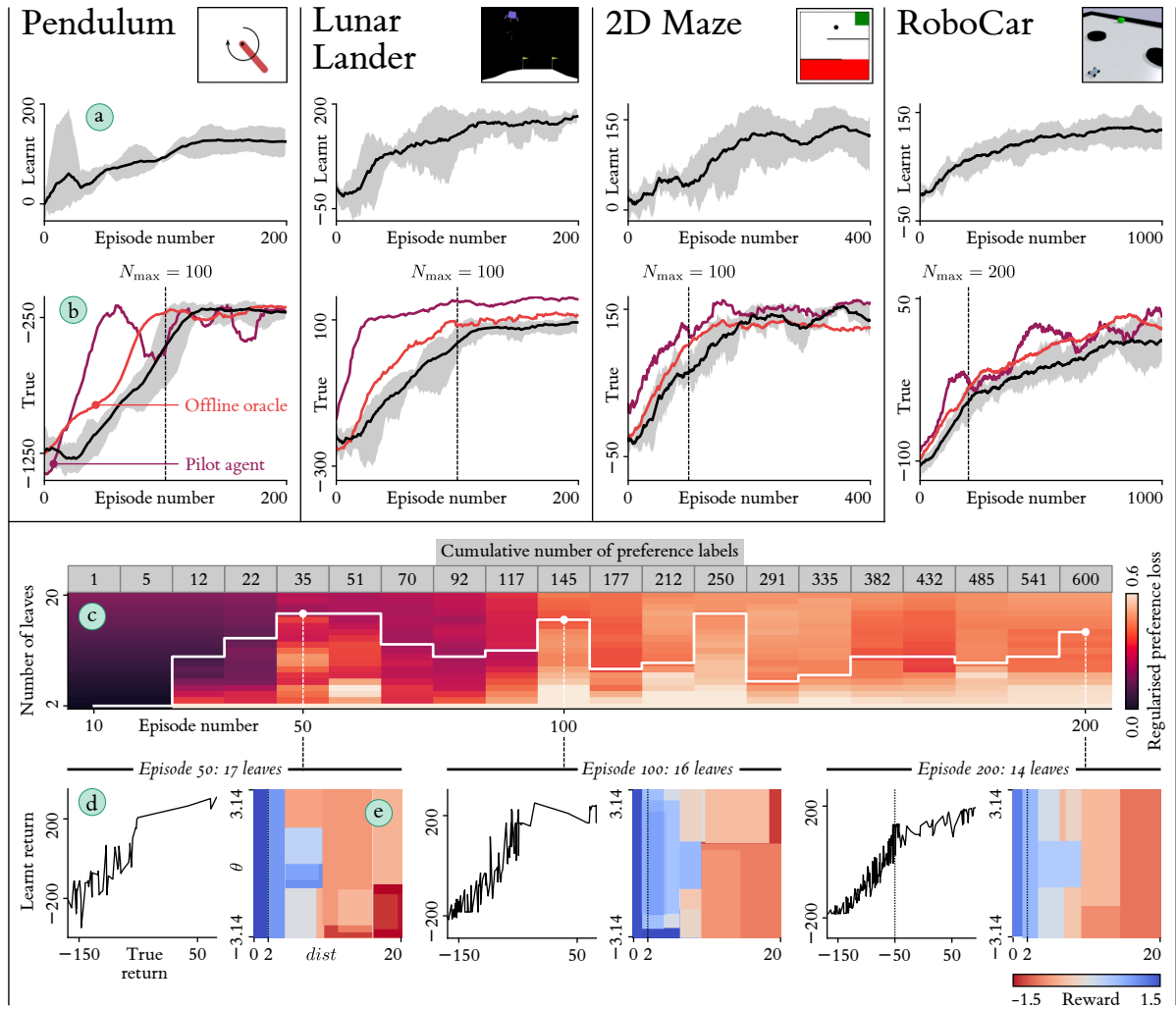


Figure 5.8: Performance in online setting using oracle preferences; learning timeline for RoboCar.

although a knee point emerges around a true return of -50 . We hypothesise that this may be due to the indicator term in the ground truth reward function (Equation 5.20), which introduces a sharp nonlinearity that is smoothed out by our averaging approach to leaf reward estimation. Leaf projection plots for the three checkpoints (e) show that the tree converges to a structure that positively rewards both proximity to the goal (small $dist$) and facing towards it ($\theta \approx 0$), doing so in an almost-symmetric manner. Again comparing to the ground truth in Equation 5.20, we note that the negative relationship to distance is correct. However, the critical threshold at $dist = 2$ is most closely replicated by the earliest checkpoint before being less accurately modelled thereafter. Additionally, the ground truth has no explicit dependence on θ . The fact that the tree has learnt to reward $\theta \approx 0$ is likely a result of correlations in the training data: trajectories where the car reaches the goal will also involve facing towards it. Although technically a misalignment, this difference may not necessarily harm agent performance, as it could provide beneficial shaping that makes goal-reaching more likely.

5.7 Quantitative Performance: Human Preferences

This section presents results using real human preferences collected from experimental participants. In order to retain a basis for quantitative evaluation, we brief participants with textual task descriptions which summarise the ground truth reward functions in Section 5.5.1 and request that they provide preference feedback on that basis. This means that the success of the end-to-end learning process can still be assessed in terms of agent performance on the ground truth reward. However, unlike the idealised oracles, humans may suffer from misinterpretation of the task description and may make random mistakes in their preference labelling, both of which make the reward learning problem more challenging. Unless otherwise stated, the hyperparameter values used in this section are the same as in our oracle experiments.

5.7.1 Offline Setting

Using the same set of pilot trajectories as in Section 5.6.1, we build a web-hosted survey to gather pairwise preference labels from human experimental participants.⁵ As shown in Figure 5.9 (a), the survey interface contains a textual summary of the current task under consideration, infinitely looping videos of the current trajectory pair (animated versions of the images in Figure 5.6) and an input field for giving preferences on a 0-10 scale. Although our algorithm interprets labels as binary (first trajectory preferred for 0-4, second preferred for 6-10, uniform random preference for 5), we use this more fine-grained numerical scale to assess participants' calibration to ground truth return differences (see below). Each participant is given 10 trajectory pairs to evaluate for each of the four tasks. Since the survey is run asynchronously with many participants, it would have been technically complex to perform tree updates on the back-end throughout the (7 day) survey period. For this reason, we wait until all responses are gathered before doing a single round of tree growth and pruning.⁶ Five independent RL training runs are then performed using the resultant reward tree for each task.

We recruit participants via a non-selective call for participation across our personal and academic networks, as well as on a public forum for sharing research surveys. The call yields 62 respondents, and thus a total preference dataset size of $K_{\max} = 62 \times 10 = 620$ for each task (very similar to the 600 used in oracle experiments). As shown in Figure 5.9 (b), the participants have a wide variety of experience levels with RL and AI more generally; almost half have no experience whatsoever. As additional background, we ask participants to indicate their expectations of the likelihood of our method succeeding, both before and after completing the survey (c), and to rank the tasks by their perceived difficulty of giving helpful feedback (d).

⁵The study design received ethics approval from the University of Bristol research ethics board (approval code: 0243). The survey page itself can be found at <https://forms.gle/cZ39tsiEr2fq8N7A6>.

⁶Since Algorithm 3 initiates with uniform optimistic return estimates, the result is that pairs are sampled uniform-randomly throughout the survey. This somewhat reduces the comparability of these results to those from the offline oracle experiment, where $K_{\text{tree}} = 60$. We may have attained better performance in this experiment if periodic updates were possible, as it would have enabled the optimistic sampling method to work as designed.

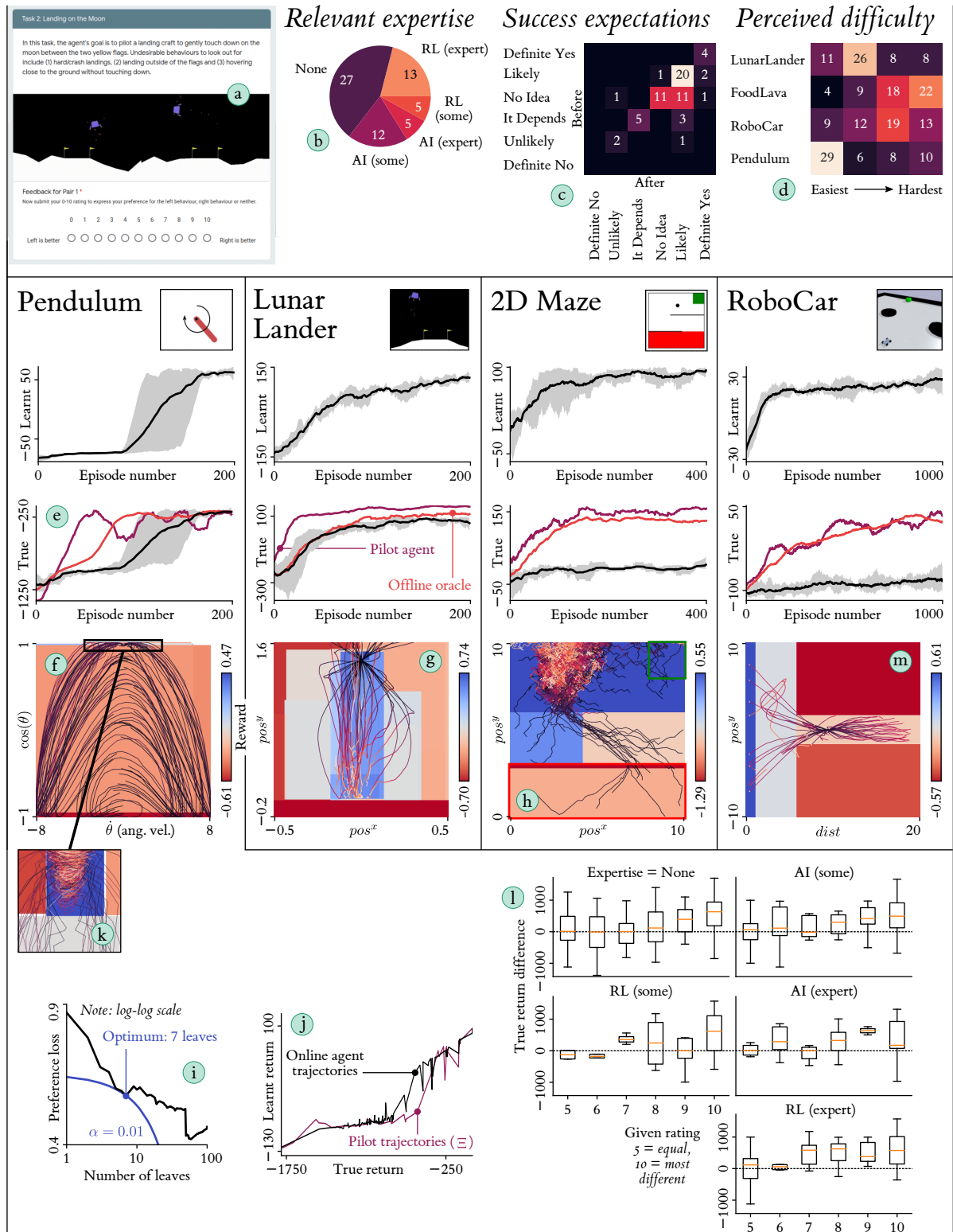


Figure 5.9: Performance in offline setting RL using human preferences; additional plots for Pendulum.

Interestingly, this ranking predicts the success of our method in this setting, since for Pendulum and LunarLander, we achieve asymptotic ground truth return (e) equal to, or only slightly below, the oracle results. This means that despite the potential error sources of task misinterpretation and preference noise (as well as possible inconsistencies between participants), reward trees are learnt that incentivise aligned RL policy learning. This is a positive result for the viability of reward tree learning in real-world contexts. The leaf projection plots (f) and (g) provide insight into the learnt tree structures. In Pendulum, high reward is only given only the pole is close to upright with small angular velocity, indicating that participants tend to be very stringent in their interpretation of ‘upright pole-balancing’. In LunarLander, high reward is given in a narrow column above the landing pad, reflecting participants’ preference for landing trajectories that do not deviate too far from the midline. The final 10 trajectories from all five RL runs are overlaid, confirming that the agents learn to seek out high-reward leaves and consequently solve the respective pole-balancing and landing tasks.

We are unable to achieve aligned learning in 2D Maze and RoboCar, although in the former, the outcome is not as catastrophic as the learning curve suggests. As shown in the leaf/trajectory plot (h), the agent learns to solve most of the maze but is not incentivised to proceed to the goal because the highest-reward leaf covers the entire upper third. This, we hypothesise, is evidence of a *causal confusion* problem [247]: within the pilot dataset, almost all trajectories that reach the upper third then go on to the goal, so the participants’ somewhat imprecise trajectory-level preferences are unable to communicate that the latter step is necessary for high return. Another contributing factor may be that participants struggle to avoid mentally extending trajectories that come close to, without actually reaching, the goal, and thus do not sufficiently penalise ‘not-quite goal reaching’ behaviour with their preferences (this phenomenon is discussed and modelled in [145]). The poor result in RoboCar can also be attributed to causal confusion, and we investigate this as part of an interpretability analysis in Section 5.9.

For Pendulum, we include the preference loss curve (i), showing the optimal pruned tree size of 7 leaves. We also plot the alignment between true and predicted returns for pilot and online trajectories (j). This curve has a knee in the upwards direction, which may be because the ground truth reward (Equation 5.17) is a smooth quadratic function while the tree only gives high reward in the small localised region visible in the inset plot (k) (this reasoning is the inverse of that given for Figure 5.8 (d)). This suggests that, relative to the ground truth, the participants may be *too* stringent in their interpretation of the balancing task, although this does not harm agent performance. Finally, we show box plots of the agreement between provided preference labels (inverted for values of 0-4) and ground truth return differences (l). Preferences from participants of all expertise levels generally align with return, as they are above the dotted line. Although the trend becomes slightly more pronounced for more certain labels (closer to 10), this is far from clear-cut, suggesting that numerical preference strengths are a less reliable signal than their binary directions. There is also a slight indication that those with RL and AI expertise exhibit somewhat lower variance than non-experts.

5.7.2 Online Setting

Finally, we perform online reward tree learning using preferences provided by a single human participant (the author of this thesis). To do this, we implement a graphical user interface to collect preference labels over trajectories generated by an RL agent running locally on the same machine. Mirroring the survey used in the offline human experiment, the interface presents trajectory pairs side-by-side as infinitely looping videos. Due to the labour-intensiveness of this experiment, we focus on two environments: 2D Maze and LunarLander.

Both cases reveal the risk of prematurely freezing the reward tree structure. We initially use $N_{\max} = 100$, as in the oracle-based experiments. Figure 5.10 (a) (dotted lines) shows that the agents quickly converge to high return according to the learnt reward functions, but (b) shows that after episode N_{\max} , this leads to a steady worsening of performance according to the ground truth reward. This indicates that the reward structure has been frozen in a state that is only partly aligned, such that further optimisation hinders, rather than helps, true performance. This phenomenon is well understood and is known as *reward hacking* [126, 232]. Leaf projection plots of the final trees for these runs show that for 2D Maze (c), the maximum positive-reward leaf is located around the goal region, but is ‘loosely’ targeted as it exceeds the true bounds of the goal. The overlaid final trajectories show that the agent learns to seek out this high reward but sometimes stops short of the goal itself. For LunarLander (d), a similarly ‘loose’ reward function is learnt that gives high reward for a vertical position close to zero, regardless of the vertical velocity. The trajectory overlay shows that this reward function leads the agent to maintain high negative velocity as it approaches the ground, which is recognised as a crash landing by the ground truth reward function.

Since this phenomenon does not arise in the online oracle experiments, it is likely due to a complicating feature of real human preference data. We hypothesise that humans may adopt a *nonstationary* feedback strategy, which initially gives a favourable preference to any trajectory that tends in a promising direction (i.e. towards the goal for 2D Maze, towards the ground for LunarLander), even if the route there is imprecise, inefficient or even unsafe as in the LunarLander crash landings. Later, as the agent’s policy begins to improve, they adapt their strategy to be more stringent about details in order to fine-tune the behaviour. There is significant prior evidence of human feedback being a moving target in this way [144]. In our case, freezing the reward trees at $N_{\max} = 100$ may be too early, giving the human insufficient time to shift to the fine-tuning regime. We note that the interpretability of reward trees, specifically the ability to visualise their leaf geometry directly, was crucial in diagnosing this issue.

With the above hypothesis in mind, we complete a second run in each environment using the same K_{\max} but a higher value of N_{\max} (300 for 2D Maze, 190 for LunarLander), thereby distributing the preference budget over a larger fraction of the agent’s training, and providing more time for the human’s feedback strategy to shift. We find that this resolves the reward hacking problem. In each second run, ground truth return (b) almost matches the pilot agent

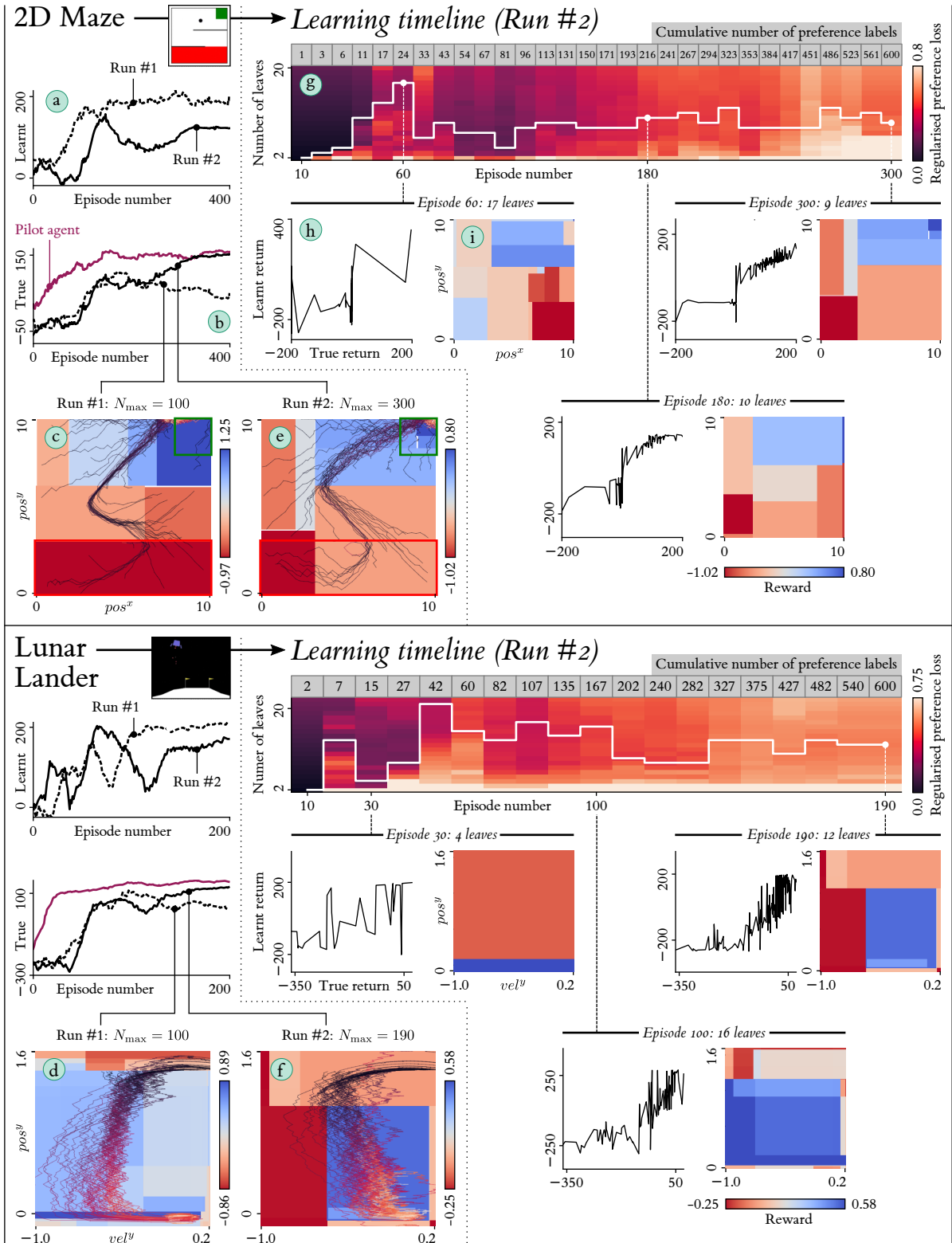


Figure 5.10: Performance in online setting using human preferences; results with learning timelines for 2D Maze and LunarLander.

from the offline oracle experiment, with no sign of a performance dropoff. For 2D Maze, the leaf projection plot (e) reveals a much smaller area of maximum reward that does not exceed the bounds of the goal region and thus incentivises the agent to enter it reliably. For LunarLander (f), we obtain a very different reward structure to the first run, which gives positive reward for maintaining slow vertical velocity and negative reward for exceeding a velocity threshold of -0.55 , rather than merely rewarding reaching the ground. As the overlaid trajectories show, this incentivises the agent to gradually decelerate as its height decreases, resulting in softer landings that are not registered as crashes by the ground truth reward function.

For both environments, we show a learning timeline for the second run (g). As in the oracle experiments, we see the trend of tree size increasing to a maximum before stabilising at a lower value. The checkpoints also show the relationship between the true and learnt return becoming less noisy over time (h) and three intermediate leaf geometries that emerge during training (i). Notably, for LunarLander, the tree at the earliest of the three checkpoints (episode 30) is similar to the final result of the first run, in that it gives high reward for reaching the ground, regardless of the vertical velocity. This suggests that the human’s velocity-agnostic early feedback strategy is consistent across training runs but evolves away given sufficient time.

5.8 Summary of Key Findings

- With no exceptions, training SAC RL agents using reward trees gives rise to stable, convergent policy learning. This remains true even as the tree is continually regrown and pruned using online preferences, making the agent’s objective nonstationary.
- Using several hundred instances of oracle preferences based on ground truth reward functions, our algorithm can reconstruct those reward functions sufficiently well to train agents whose performance nearly matches that of agents with direct ground truth access.
- In most cases, the aggregated preferences of 62 human participants (offline), as well those of a single participant (online), yield learnt reward trees that are similarly well-aligned to the ground truth. This is despite participants having no direct knowledge of the ground truth, instead relying on their intuitive understanding of textual task descriptions.
- In the offline setting, dataset biases can lead to causal confusion, where learnt reward trees incentivise transitions that commonly appear alongside high-reward behaviours, as well as the behaviours themselves. Careful rebalancing of training data, or moving to an online learning setup, may help to mitigate this problem.
- When learning from real human preferences in the online setting, the main failure mode is reward hacking due to freezing the reward tree prematurely in a ‘loosely’ aligned state reflecting the human’s temporary early feedback strategy. Increasing N_{\max} gives more time for the human to shift to a fine-tuning strategy.

5.9 Interpretability Analysis

The analyses of leaf projection plots and learning timelines in Sections 5.6 and 5.7, which provide insight into the learnt reward functions and their effect on RL agent behaviour, are an initial hint at the interpretability benefits of using tree-structured models for reward learning. They have enabled us to diagnose the sources of misalignment in some cases, verify success in others, and generally build an understanding of how our algorithm operates. In this section, we make the interpretability of reward trees our primary focus and present a selection of other visualisations and analyses that are made possible by their adoption. Favouring depth over breadth, we narrow our scope to two specific reward learning runs from the preceding experiments: one failure case and one success case.

5.9.1 Failure Case: RoboCar using Offline Human Preferences

This reward tree was learnt from the aggregated preferences of the 62 survey participants. As is visible in Figure 5.9 (m), it leads to policies that sometimes reach the goal as desired, but other times make no progress towards the goal and appear to seek only to maintain a vertical position close to zero. We can diagnose this misalignment by examining the reward tree diagram, shown in Figure 5.11 (a), where leaves are coloured by their reward predictions.

Here, the features used for splitting are vertical position pos^y , distance to goal $dist$, and bearing to goal in radians θ . The mostly-symmetric treatment of pos^y and θ about zero is itself correct because the environment has inherent symmetry in these dimensions. The first two splits also appear well-aligned, creating a leaf with maximum reward for achieving $dist < 1.16$ (leaf 1) and a smaller positive reward for $dist < 5.84$ (leaf 2). However, the remaining splits are problematic, creating leaves that penalise moving out of the region $pos^y \in [-1.64, 1.68)$ and, otherwise, reward a bearing outside of $\theta \in [-2.15, 2.18)$ (i.e. facing *away* from the goal). To understand this, consider the design of the environment. In each episode, the goal position is randomised, but the car is initialised at $pos^y = 0$ and facing to the right, making it easier to reach the goal when it is also to the right. Hence, many goal-reaching trajectories in the pilot dataset (especially those from early in the pilot’s learning) show the car driving directly forward, rarely exiting a narrow corridor around $pos^y = 0$. The splits to penalise large absolute pos^y are thus an example of causal confusion, in which behaviour correlating with a high-reward outcome is mistaken for deserving high reward in itself, and would likely not appear if the environment were differently initialised or the dataset better balanced.

We can give a similar, if subtler, causal confusion interpretation of the θ -based splits. Of these, the first two splits create positive-reward leaves for $\theta \geq 2.18$ or $\theta < -2.15$ and negative reward otherwise. An agent is thus rewarded for facing *away* from its goal, which is heavily misaligned. The near-exact symmetry of these two splits implies that they are also due to a reliable feature of the environment rather than a statistical fluke, and our diagnosis is as

follows. In RoboCar, the agent must learn to navigate the car around four obstacles to reach the goal. In the pilot dataset, many successful goal-reaching trajectories feature the car initially moving away from the goal to bypass an obstacle. As a result, many transitions in trajectories with favourable human preferences fall outside of the $\theta \in [-2.15, 2.18)$ region. Meanwhile, many non-preferred trajectories involve the car driving directly at the goal and becoming stuck against an obstacle for many timesteps in a row. Thus, if the car is *both* far from the goal (i.e. $dist \geq 5.84$) *and* facing towards it ($\theta \in [-2.15, 2.18)$), it is statistically more likely that this transition is a member of a low-return trajectory than a high-return one. The opposite is true for timesteps spent facing away from the goal. This leads our algorithm to fall foul of causal confusion by creating a tree that rewards facing away from the goal, a circumstance which merely correlates with high true return (in the pilot dataset) rather actively than driving it.

An aligned solution to this problem would be to split on the obstacle contact indicator feature c^o instead, creating a leaf that penalises collisions directly. It is not entirely clear why this option is not taken, but it may be that the human survey participants are not sufficiently consistent in their treatment of obstacle contact in their responses. Moving to an online learning setup may help to reduce the likelihood of such confusion persisting, as it provides an opportunity for the human(s) to reactively penalise early examples of behaviour that resulted from it. We are unable to meaningfully diagnose the final asymmetric split at $\theta = -0.676$, which suggests that this is likely due to a random statistical imbalance in the pilot dataset.

The heatmaps [\(b\)](#), [\(c\)](#) and [\(d\)](#) provide fine-grained insight into the effect of the misaligned reward tree on the learning dynamics of one of the five RL training runs. [\(b\)](#) represents the number of timesteps spent in each leaf over the 1000-episode training history. Scaling this matrix by leaf rewards, we obtain [\(c\)](#), which gives per-episode return from each leaf, and can be understood as a decomposed learning curve. Summing [\(c\)](#) column-wise gives [\(d\)](#), the total return for each episode, which equates to a conventional learning curve. From these, we find that the agent quickly (by episode 50) learns to avoid negative-reward leaves 3 and 8, inducing an early bias towards exiting the $pos^y \in [-1.64, 1.68)$ corridor. With this bias in place, exploration is curtailed, and the agent largely settles into the moderate positive rewards of leaves 4 and 7. Although there is a gradual increase in visitation to leaf 1 (the one corresponding to reaching the goal) in the first half of training, the agent never completely prioritises this leaf, with visitation peaking around episode 700 before dropping off again. In [\(e\)](#), we harness the tree’s rule-based structure to construct textual *report cards* for two episodes near the end of training (950 and 975), which use rules to describe the subsets of feature space that are visited. While both are in the top 10% of episodes by performance on the learnt reward, the former is aligned (obtaining positive reward from leaves 1 and 2) while the latter is not (staying entirely in leaf 6, thereby driving straight ahead despite the goal being behind it).

In summary, this subsection shows how the rule-based structure of a reward tree, and the implicit reward decomposition that it induces, allow us to identify both the causes of

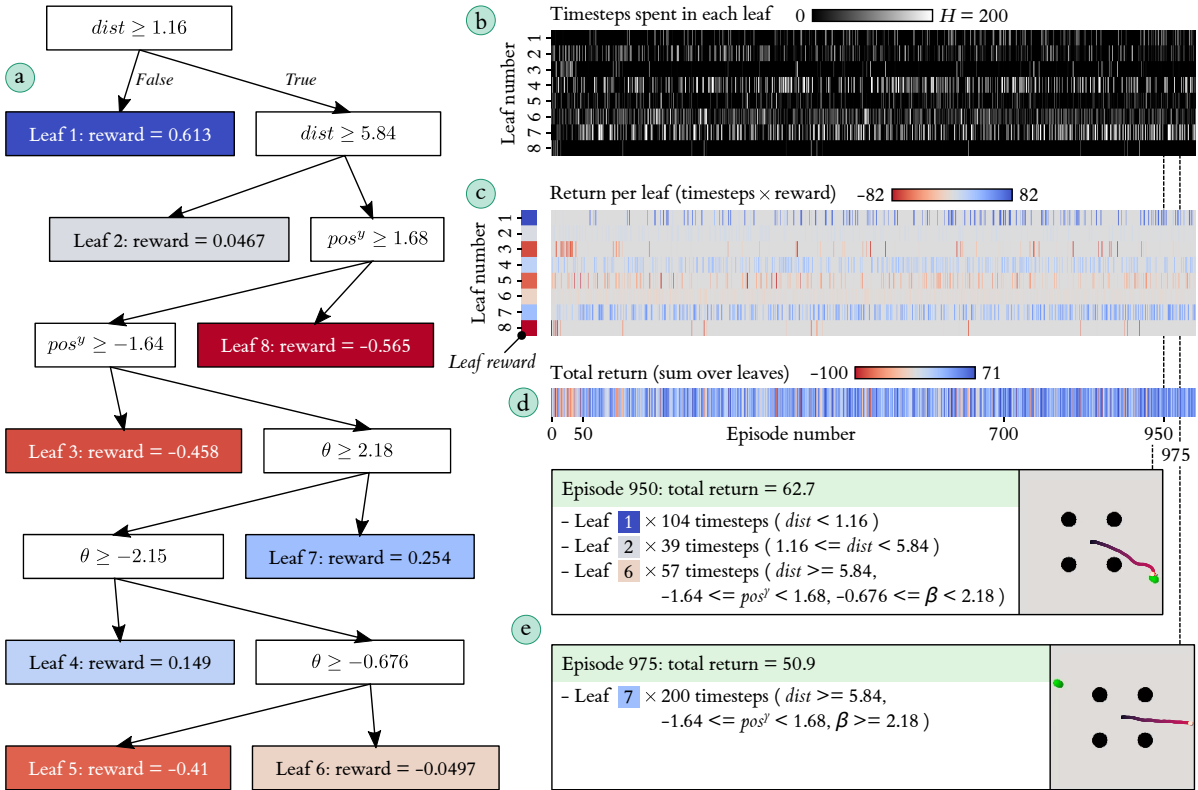


Figure 5.11: Tree diagram and other visualisations for a failure case in RoboCar.

misalignment and their effects on individual episodes and long-term learning trends. In the particular case analysed, correlations in the offline dataset create a causal confusion effect, leading the reward tree to perversely incentivise keeping the car in a horizontal corridor, and even driving away from the goal location. It is important to note that traditional reward learning models, including neural networks, are susceptible to causal confusion [247], but their uninterpretable structures may make this harder to identify. Hence, even in failure, reward tree learning can offer significant benefits for verification and validation.

5.9.2 Success Case: 2D Maze using Online Oracle Preferences

In this run (randomly selected from the five repeats), we achieve aligned reward and policy learning. For the first $N_{\max} = 100$ episodes, a batch of preference labels is obtained at intervals of $N_{\text{batch}} = 10$, and the tree structure is updated by growth and pruning. Hence, 10 updates are performed in total. Figure 5.12 (a) depicts the net changes resulting from each update using both leaf projection plots and a directed graph of the split/prune dependencies between leaves from update to update. Note that the graph does not represent all split and prune operations made within each update step, but rather the aggregate change between the tree structures before and after the update. These plots reveal key events in the iterative learning of the reward tree, which may be of value to model developers and other technical experts.

Such events include the pruning of four leaves in update 4, which indicates that these leaves are no longer sufficiently beneficial for preference loss reduction to outweigh the increased regularisation penalty. This pruning step removes a misaligned leaf, which had given high reward for entering the top-left corner of the maze, and may have negatively impacted agent learning if left unpruned. Also of note is the corrective splitting, pruning, and re-splitting (at a different threshold) of a leaf between updates 8 and 10, yielding the final maximum-reward leaf 9 whose hyperrectangle boundaries (horizontal position $pos^x \geq 7.95$, vertical position $pos^y \geq 8.06$) line up almost exactly with the goal region ($pos^x \geq 8$, $pos^y \geq 8$). This example demonstrates how gathering more preferences enables more accurate reconstruction of a ground truth reward function, and how iterative tree updates can undo and correct previous split decisions to improve alignment. In update 6, the tree structure is left entirely unchanged (i.e. the optimal pruned tree is identical to the one from which growth is started). However, the reward assigned to each leaf is updated to reflect the latest set of trajectory return estimates.

From update 2 onwards, leaf 1 ($pos^x < 7.95$, $pos^y < 3.83$, which covers most of the penalty region) is persistent, being neither split nor pruned. However, its reward estimate is continually refined as preference labels arrive, a process visualised in (b). For each update, every trajectory ξ^i that has been labelled so far (of which there are more for later updates) is shown as a black horizontal line, whose vertical position corresponds to its trajectory-level return estimate g_i and whose length is proportional to the time spent in leaf 1. The leaf reward estimation method described in Section 5.3.2 effectively computes a length-weighted average over these line positions, and the results for all updates are overlaid. The shaded area corresponds to the square root of the leaf’s impurity at each update, which is also the standard deviation of the reward estimate itself. Overall, the reward remains relatively stable over time, even as new trajectories arrive, and new preferences change the estimated returns of existing ones. This indicates that for this particular leaf, relatively few preferences are required to obtain a good reward estimate. That said, the period between updates 2 and 10 sees a slight increase in the estimate, and a gradual reduction in the impurity, as more trajectory pairs are labelled.

(c) depicts the diagram of the frozen 9-leaf tree used for the final $N_{\text{conv}} = 300$ training episodes, and (d) shows the timesteps spent in each leaf throughout this period. As of episode 100, roughly equal time is spent in leaf 2 (medium reward; covers most of the upper part of the maze) and leaf 9 (high reward; corresponds almost exactly to the goal region). However, the latter comes to dominate around episode 250, indicating that from this point onwards, the agent consistently and rapidly solves the maze navigation problem. The occasional failures near the end of learning occur when the agent spends an unusual amount of time in leaf 1 (highlighted as (e)). This is likely due to it becoming stuck underneath the lower wall of the maze. Note that this visualisation provides similar insight to the marginal abstract state visitation plots in Chapter 4 (e.g. Figure 4.8 (b)), but has additional semantics because of how reward tree leaves directly determine the reward that the agent receives.

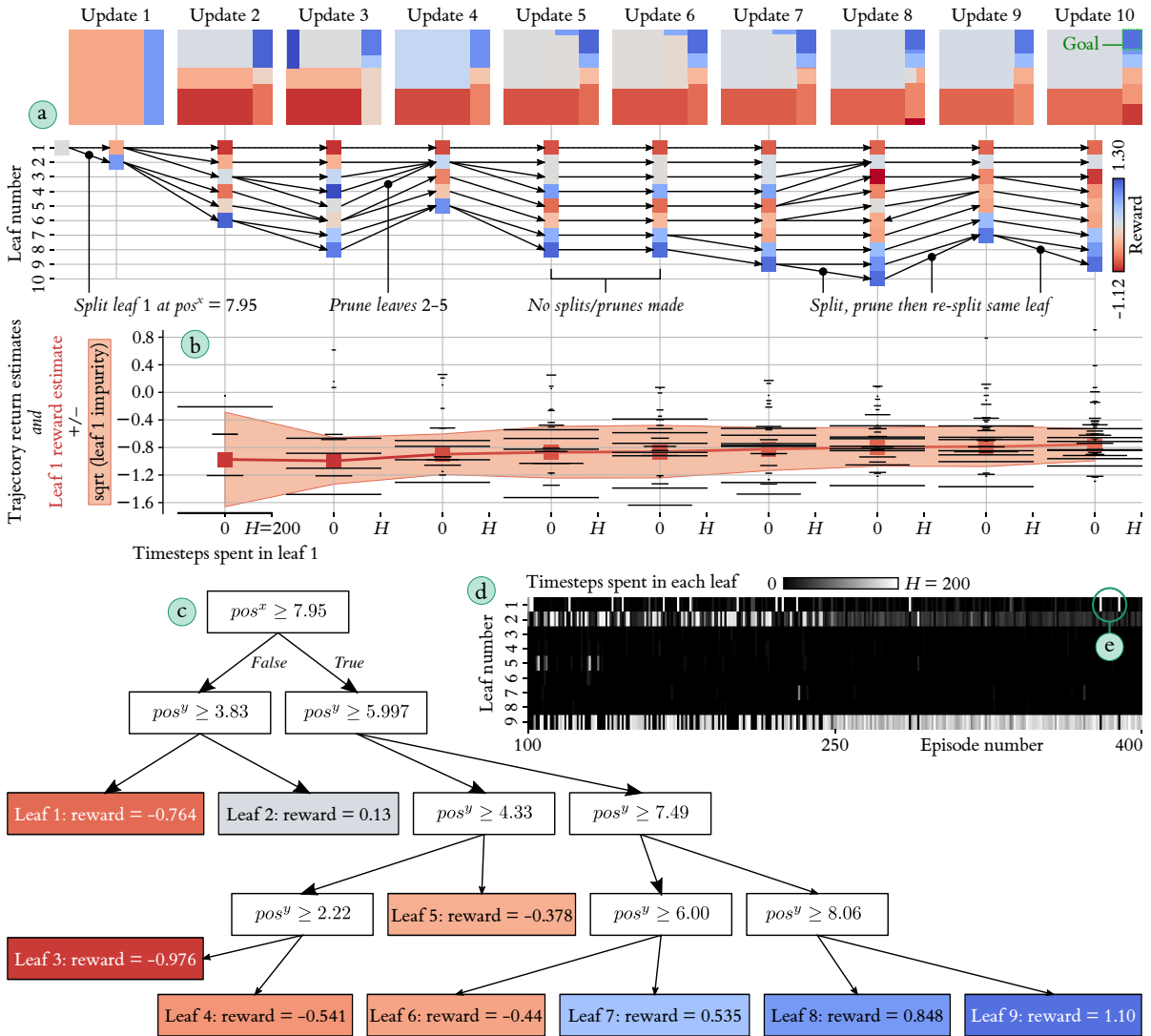


Figure 5.12: Tree diagram and other visualisations for a success case in 2D Maze.

In summary, while the preceding subsection explored the value of reward tree interpretability for diagnosing failures, this subsection shows how it helps us to verify success by monitoring the temporal interactions between agent and reward learning. The two novel visualisations (a) and (b) deliver interpretability not merely of the final tree structure, but also of the algorithmic process by which it is generated, and of the changes in agent behaviour that it induces. An extension of this analysis would be to attribute changes in tree structure, such as those depicted in (a), to the arrival of individual preferences. This may highlight when preferences are especially influential or inconsistent. When learning from real human preferences, it may provide a route to understanding temporal changes in the human’s feedback strategy, of the kind discussed in Section 5.7.2. In the next chapter, we propose a preliminary method for measuring the influence of individual preferences on reward tree predictions.

5.10 Conclusion

In this chapter, we have taken a perspective on agent interpretability that differs from earlier chapters. We began by noting the growing interest in the reward alignment problem, the popularity of human-in-the-loop approaches to tackling it, and the lack of prior emphasis on interpretability in this area. From this starting point, we proposed to use trees to learn reward functions from human preference feedback. These reward trees provide an interpretable bridge between human preferences and agent behaviour, enabling each to be described and explained in terms of the other. Building on the notational and algorithmic tools established in earlier chapters, we described a novel method that learns a reward tree from preferences and (subsequently or concurrently) trains an RL agent with the tree as its objective. In experiments with real and synthetic human preference data, we demonstrated successful learning of compact and well-aligned reward trees across four environments, alongside informative and actionable failure cases due to causal confusion in the offline setting and premature reward freezing (paired with human nonstationarity) in the online setting. We then showed the value of interpretability for exploring and debugging the learnt reward structure. The models and experiments in this chapter have several limitations, which create opportunities for further work:

- Our current method learns only from pairwise trajectory preferences, which are popular in the literature, but it would be valuable to generalise it to multiple kinds of feedback signal such as manual demonstrations, state- or trajectory-level approval labels, or local action corrections. We suspect that implementing this multimodality is tractable because all such feedback modes can be given a common mathematical formalism [131].
- Our human experiments assessed the method’s ability to learn from real user preferences but did not ‘close the loop’ to see how those same users would interpret and evaluate the resultant reward trees. In particular, it would be informative to see if non-experts could identify flaws and omissions in reward tree structures, and correct them through targeted (potentially multimodal) feedback.
- Although promising results have been obtained, the current reward tree growth and pruning stages could likely be refined to improve performance. In the next chapter, we adopt a new split criterion that directly works to minimise the preference loss.
- While our experiments considered various environments and tasks, they were all relatively simple benchmarks. It is important to assess the scalability of reward tree learning to more complex, high-dimensional problems. This is also a focus of the next chapter.

Finally, we wish to highlight the work of Kalra and Brown [135, 136], which explicitly builds on our own to learn differentiable reward trees from trajectory preferences. These models use general (i.e. non-axis-aligned) hyperplane rules with ‘soft’ activation functions that route inputs down all branches of the tree with nonzero probability. As a result, they are more expressive

than our reward trees and may require less feature engineering to achieve good performance. Early results are promising. However, differentiable trees require a fixed number of leaves to be specified a priori rather than adapting in size via growth and pruning. Their complex parameter structure and probabilistic prediction mechanism also make them significantly harder to interpret. Since different use cases have different priorities in the tradeoff between performance and interpretability, this proposal for a semi-interpretable model is welcome.

Chapter 6

A Use Case for Reward Trees

Based on: “Learning Interpretable Models of Aircraft Handling Behaviour by Reinforcement Learning from Human Feedback”, accepted for presentation at 2024 AIAA SciTech Conference.

6.1 Introduction

The preceding chapter established the basic efficacy of preference-based reward tree learning in simple environments, including popular benchmarks. In this chapter, we apply the method to a more challenging, industrially-motivated use case in the aviation domain. In the process, we make methodological changes which improve performance, and develop a more extensive pipeline for quantitative and qualitative evaluation. We also explore additional directions for exploiting the interpretability of the reward tree architecture. As a result, this chapter makes a number of contributions beyond those of the preceding one:

- The identification of a realistic pathway for interpretable reward learning to address a concrete industrial need. While this specific use case happens to be in the aviation domain, we suspect that it is representative of a more general class of cross-industry applications.
- Several targeted changes to bring our method more in line with prior reward learning work and improve learning outcomes. In particular, we find that adopting a novel split criterion during tree growth significantly improves quantitative and qualitative performance.
- The integration of reward tree learning with a model-based RL agent, which greatly accelerates learning and enables a mechanism for explaining action selection that is synergistic with the reward tree architecture.
- A more thorough evaluation pipeline, involving a range of performance metrics that can be compared across tasks, a qualitative review of learnt behaviours, and a sensitivity analysis with respect to data availability and biased or noisy preferences. Throughout, the performance of reward trees is baselined against the de facto standard approach of reward learning using neural networks.

- In light of this evaluation, positive empirical evidence that reward learning can be done effectively using interpretable tree models, even in complex, high-dimensional continuous environments. Trees are found to be broadly competitive with neural networks on both quantitative metrics and qualitative assessments, and exhibit equal or better robustness to limited or corrupted preference data.
- A further exploration of interpretability methods, including a novel way of combining reward trees and model-based RL to explain agent action selection.

The suite of evaluations in this chapter require a large quantity of preference data for several tasks in our simulated aviation domain. Since collecting such data from real pilots and aviation experts would be costly and logistically complex, all experiments use synthetic preferences with respect to nominal oracle evaluator functions of varying complexity. As noted in the previous chapter, using oracles as a proxy for human evaluators is popular because it enables the definition of ground truth performance metrics. However, emulating a human with an oracle that responds with perfect rationality is unrealistic [153]. It is for this reason that we move beyond the approach of the previous chapter to examine the performance impacts of noisy and myopic oracles in the sensitivity analysis. Nonetheless, an evaluation with real human experts is an important direction for future work.

6.2 Aircraft Handling Use Case

6.2.1 Motivation

Pilots of fast jet aircraft require exceptional handling abilities, acquired over years of advanced training. There would be significant practical value in a method capable of distilling the skills, knowledge and preferences of pilots and other domain experts into a software model that captures realistic handling behaviour. The scalability of such a model would make it useful for strategic planning exercises, training, and development and testing of other software systems. This would enable greater return from the scarce resource of human piloting expertise.

This vision is bottlenecked by the practical challenge of accurately eliciting the desired knowledge for codification into an automated system [56]. As in many contexts requiring intuitive decision-making and rapid motor control, the preferences of experts about what constitutes safe and effective aircraft handling behaviour are in large part tacit, and thus defy direct scrutiny or exact verbal description [236]. Put simply: experts know good handling when they see it, but cannot always express *why* in formal or linguistic terms. Even in cases where preferences do appear to be explicit (e.g. industry-standard heuristics with a rule-like form), it is debatable whether these simple generalisations should always be adopted when implicit expertise may suggest otherwise. Such contradictions are especially likely to arise in rare and dangerous edge cases, and blind adoption of the presumed explicit knowledge into the objective function of an automated system risks inducing a paradigmatic example of misalignment. An explicit

knowledge elicitation strategy would also likely be time-intensive, as would any approach relying on expert demonstration. This motivates a learning-based approach to infer the structure of implicit expert preferences from a sparser data source.

In light of the importance of transparency for safety-critical aviation applications [35, 214], it is crucial that any such approach learns an interpretable model of the expert knowledge, to facilitate trust and verification. As we argued in the previous chapter, neglecting interpretability would be problematic for any realistic application of learning from humans, and this is especially true in an industry with such a justifiably strong culture of safety. Under current working practices and regulatory frameworks, an uninterpretable black box model, such as a deep neural network, would stand a poor chance of widespread adoption.

Consider how preference-based reward tree learning can provide a possible solution to the preceding brief. Assuming the online context, we use an artificial RL agent to generate a dataset of simulated fast jet flight trajectories, then consult an expert to obtain pairwise preferences over those trajectories, indicating which is preferred as a solution to a given task of interest (e.g. turning to follow another aircraft, or landing on a runway). We then use our multi-stage learning algorithm to construct an interpretable explanatory model of the gathered preferences in the form of a rule-based tree structure. In turn, the tree is used as a reward function to train the agent to generate higher-quality trajectories, and the process is iterated to convergence. The end result is two distinct outputs that could form valuable components of future planning, training and development software:

1. A reward tree, which captures tacit expert preferences over fast jet handling behaviours in a human-readable form. The tree may be used for consistent automated scoring of flight trajectories executed by human or artificial pilots in a way that is aligned with the judgement that the original expert would have made, alongside an explanatory rationale that can be used to justify, verify and improve handling behaviour.
2. An RL agent capable of executing high-quality handling with respect to the objective specified by the reward tree, for use in simulation (e.g. as a demonstrator for pilot training).

Driven by this strong motivation, this chapter uses fast jet handling as a representative case study for how reward tree learning may be used to address a concrete industrial need.

Prior work has seen widespread adoption of RL for aviation [15, 164, 206]. It has been used to learn landing [243] and aerobatics [48] behaviours for fixed-wing aircraft, as an alternative or supplement to learning from costly human demonstrations [36, 181]. Other work has retained a focus on learning from and with humans, using RL to predict pilot interactions in an airspace [270], implement shared autonomy for single aircraft control [259], and simulate student learning dynamics in pilot training [255]. As in most other contexts, aviation applications of RL have typically required rewards to be heuristically (thus potentially erroneously) defined rather than flexibly learnt, and lack interpretability of the underlying model. Our approach is a novel integration of humans into the RL process that is both effective and intrinsically interpretable.

6.2.2 Implementation

To formalise the fast jet handling problem in a way that is compatible with RL and reward learning, we consider a simple set piece setup, in which the piloting agent is given a short time window (i.e. an episode) to manoeuvre its aircraft (the *ego jet*, EJ) in a particular manner relative to a second *reference jet* (RJ) whose motion, if any, is considered part of the environment dynamics. The state space \mathcal{S} contains the positions, attitudes, velocities and accelerations of both EJ and RJ, and the action space \mathcal{A} consists of pitch, roll, yaw and thrust demands for EJ only. The EJ dynamics function integrates these demands with a simplified physics engine, including gravity and air resistance. RJ dynamics, as well as the conditions of episode initialisation and termination, vary between tasks as described below.

This set piece formulation strikes a balance between simplicity and generality; many realistic scenarios faced by a fast jet pilot involve interaction with a single other airborne entity. It provides scope to define many alternative tasks given the same state and action spaces and largely unchanged dynamics. In this work, we consider the three tasks shown in Figure 6.1:

- **Follow:** RJ follows a linear horizontal flight path at a constant velocity, which is oriented opposite to the initial velocity of EJ. The task is to turn onto and then maintain the flight path up to the episode time limit of 20 timesteps (≈ 20 seconds, as timesteps are at approximately 1Hz). This constitutes a very simple form of formation flight.
- **Chase:** RJ follows an erratic trajectory generated by random control inputs, and the task is to chase it, maintaining a target distance and line of sight, without taking EJ below a safe altitude. Episodes terminate after 20 timesteps.
- **Land:** The task is to execute a safe approach towards landing on a runway, where RJ represents the ideal landing position (central, zero altitude, slight upward pitch). EJ is initialised at a random altitude, pitch, roll and offset, such that landing may be challenging but always physically possible. An episode terminates if EJ passes RJ along the axis of the runway, or after 25 timesteps otherwise.

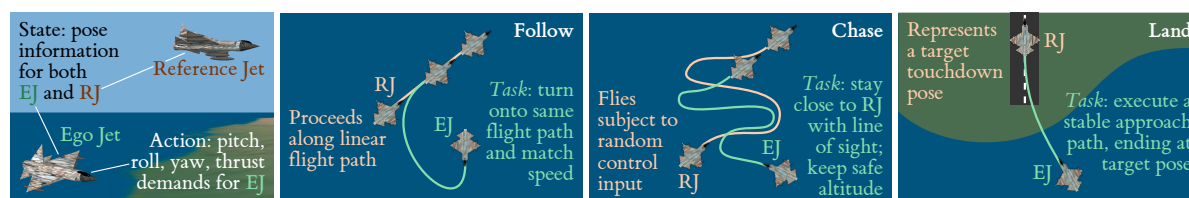


Figure 6.1: State-action space of aircraft handling domain, and diagrams of all three tasks.

The central thesis of the present work is that there exists no unambiguous model of good aircraft handling behaviour for such tasks. For instance, experts may agree that the Chase task involves a tradeoff between speed of response and smoothness of flight, but may also have

nebulous and divergent definitions of these properties and their relative importance. However, to quantitatively evaluate our method, we retain the artificial construct of synthetic oracles based on ground truth reward functions, as proxies for real human evaluators. The oracles refer to their reward functions when providing evaluative preference feedback to our reward learning method, as described in the previous chapter. The precise nature of the oracle reward functions is somewhat arbitrary, and those given below are among many equally plausible alternatives, but we dedicated several hours of development time to ensuring they incentivise reasonable behaviour upon visual inspection. The difficulty and subjectivity of such a manual reward design process is precisely why reward learning (ultimately from real human preferences) is a compelling proposition. The oracles are all defined using a common set of transition-wise features $\phi(s_{t-1}, a_{t-1}, s_t) = \mathbb{R}^D$, which are enumerated and described in Table 6.1 (overleaf):

- **Follow:** The oracle prioritises closing the distance between EJ and RJ, and matching their upward axes:

$$R_{\text{follow}} = -(\text{dist} + 0.05 \times \text{closing speed} + 10 \times \text{up error}).$$

- **Chase:** The oracle prioritises keeping RJ at a distance of 20 and within EJ’s line of sight, while keeping EJ oriented upright. It also has a large penalty for dropping below a safe altitude of 50 (recall that $[\cdot]$ is Iverson bracket notation for the indicator function):

$$R_{\text{chase}} = -(\text{abs}(\text{dist} - 20) + 10 \times \text{los error} + 5 \times \text{abs roll} + 100 \times [\text{alt} < 50]).$$

- **Land:** The oracle for this task is the most complex, including terms that incentivise continual descent, penalise g-force and engine thrust, and punish EJ for contacting the ground before the runway $[\text{alt} < 0.6]$:

$$\begin{aligned} R_{\text{land}} = & -(0.05 \times \text{abs lr offset} + 0.05 \times \text{alt} + \text{hdg error} + \text{abs roll} \\ & + 0.5 \times \text{pitch error} + 0.25 \times (\text{yaw rate} + \text{roll rate} + \text{pitch rate}) + 0.1 \times \text{g force} \\ & + 0.025 \times \text{thrust} + 0.05 \times \text{delta thrust} + [\text{delta dist hor} > 0] \\ & + 2 \times [\text{delta alt} > 0] + [\text{abs lr offset} > 10] + 10 \times [\text{alt} < 0.6]). \end{aligned}$$

6.3 Methodological Improvements

At the point of conception of this use case, our method for reward tree learning was identical to that described in the previous chapter. However, over the course of this work, several methodological changes have been found to either increase performance on the aircraft handling tasks, improve computational efficiency or bring our model into closer alignment with mainstream reward learning frameworks. The fact that the incentive to improve our method arose once we started to explore this use case illustrates one benefit of investigating larger scale problems in addition to typical benchmarks. This section describes and justifies the changes. Any aspects of the original method that are not revisited in this section have been left unchanged.

Table 6.1: Features used by oracles and reward learning models. Apart from those with “delta” or “rate”, all features are computed over the latter of the two states in each transition, s_t .

dist	Euclidean distance between EJ and RJ
closing speed	Closing speed between EJ and RJ (negative = moving closer)
alt	Altitude of EJ
alt error	Difference in altitude between EJ and RJ (negative = EJ is lower)
delta alt error	Change in alt error between s_{t-1} and s_t
dist hor	Euclidean distance between EJ and RJ in horizontal plane
delta dist hor	Change in dist hor between s_{t-1} and s_t (negative = moving closer)
pitch error	Absolute difference in pitch angle between EJ and RJ
delta pitch error	Change in pitch error between s_{t-1} and s_t
abs roll	Absolute roll angle of EJ
roll error	Absolute difference in roll angle between EJ and RJ
delta roll error	Change in roll error between s_{t-1} and s_t
hdg error	Absolute difference in heading angle between EJ and RJ
delta hdg error	Change in hdg error between s_{t-1} and s_t
fwd error	Angle between 3D vectors indicating forward axes of EJ and RJ
delta fwd error	Change in fwd error between s_{t-1} and s_t
up error	Angle between 3D vectors indicating upward axes of EJ and RJ
delta up error	Change in up error between s_{t-1} and s_t
right error	Angle between 3D vectors indicating rightward axes of EJ and RJ
delta right error	Change in right error between s_{t-1} and s_t
los error	Angle between forward axis of EJ and vector from EJ to RJ (measures whether RJ is in EJ’s line of sight)
delta los error	Change in los error between s_{t-1} and s_t
abs lr offset	Magnitude of projection of vector from EJ to RJ onto RJ’s rightward axis (measures left-right offset between the two aircraft in RJ’s reference frame)
speed	Airspeed of EJ
g force	Instantaneous g-force experienced by EJ
pitch rate	Absolute change of EJ pitch between s_{t-1} and s_t
roll rate	Absolute change of EJ roll between s_{t-1} and s_t
yaw rate	Absolute change of EJ yaw between s_{t-1} and s_t
thrust	Instantaneous thrust output by EJ engines
delta thrust	Absolute change in thrust between s_{t-1} and s_t

6.3.1 Trajectory-Level Return Estimation

In the previous chapter, we define the proxy objective for this stage as:

$$(6.1) \quad \operatorname{argmin}_{\mathbf{g} \in \mathbb{R}^N} \left[\sum_{(i,j) \in \mathcal{L}} \ell(\Pr(\xi^j \succ \xi^i | \mathbf{g})) \right], \quad \text{where} \quad \Pr(\xi^j \succ \xi^i | \mathbf{g}) = f(\mathbf{g}_j - \mathbf{g}_i).$$

We then go on to adopt Thurstone’s Case V preference model $f(z) = \Phi(z)$ and Mosteller’s least squares loss $\ell(p) = (f^{-1}(1 - \varepsilon) - f^{-1}(p))^2$. Although this loss function enables an exact solution via matrix inversion, early experiments in the aircraft handling environment found that it introduces a ‘squashing’ bias to the return estimates of the most and least preferred trajectories in a preference dataset (i.e. those with the most extreme ground truth returns, assuming noise-free preferences). The bias may be especially strong in this environment, where extremely poor trajectories (e.g. crash landings) are possible. A toy example of this phenomenon

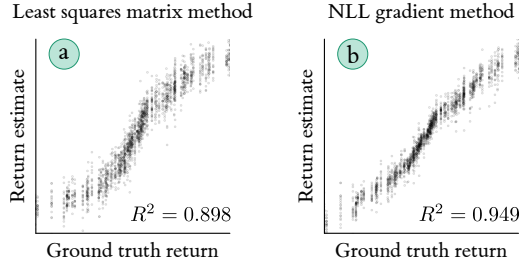


Figure 6.2: Comparison of old (least squares matrix) and new (NLL gradient) methods of trajectory-level return estimation for a toy example of $K_{\max} = 1000$ noise-free oracle preferences over $N_{\max} = 100$ trajectories with normally distributed ground truth returns. 20 repeats completed; all results plotted as scatter points.

is visible in the sigmoidal profile of Figure 6.2 (a).

Recall that the least squares loss encourages each preference in the dataset to be predicted with probability $1 - \varepsilon$ ($= 0.9$, with $\varepsilon = 0.1$). This can be perverse because it penalises probabilities higher than this value, even between trajectory pairs whose ground truth returns lie on opposite ends of the distribution in the dataset. As a result, the estimates for such extreme trajectories tend to be pulled closer towards the mean than they are on the ground truth.

The above leads us to hypothesise that the squashing bias can be mitigated by a loss function that (unlike the least squares loss) is monotonically non-increasing in predicted preference probabilities. A natural choice for this is the negative log-likelihood $\ell(p) = -\log(p)$. No exact matrix method exists for this loss, but it can be approximately minimised by gradient-based optimisation (we use the Adam optimiser [142]). As shown by the example in Figure 6.2 (b), switching to this loss function and optimisation method leads to both reduced sigmoidal squashing and a narrower local spread in the return estimates, in this case yielding an increase in the coefficient of determination from 0.898 to 0.949. In light of this reduction in both bias and variance, we adopt this method for trajectory return estimation throughout this chapter.

Additionally, we switch from the Thurstone link function $f(z) = \Phi(z)$ to the Bradley-Terry model [30], which uses the logistic function $f(z) = 1/(1 + \exp(-z))$. Aside from being faster to compute (Φ is a special function, requiring complex numerical approximation), this change has a minimal effect as the two models usually give near-identical results [128]. Our main reason for adopting it is theoretical consistency with prior work, which almost exclusively uses both the Bradley-Terry model and the negative log-likelihood loss [44, 126, 154].

Finally, we post-process the vector of return estimates \mathbf{g} to have unit standard deviation and a consistent sign (i.e. all values are either non-negative or non-positive). The downstream effect is for leaf-level reward estimates to have magnitudes on the order of 1, with the same consistency of sign. This has no impact on predicted preference orderings, which are invariant to positive affine transformations of this kind. However, we find it brings two distinct benefits:

- Preventing perverse incentives for RL agents trained with reward trees to terminate or elongate episodes in tasks with termination conditions (negative rewards on non-terminal

transitions incentivise termination, while positive rewards incentivise elongation).

- Simplifying the manual interpretation of tradeoffs between rewards from different leaves of a tree (understanding the relative impacts of “more of a negative reward” and “less of a positive reward” requires the awkward mental juggling of negatives).

For the aircraft handling task with a termination condition (Land), we use negative rewards (max = 0 constraint) to disincentivise episode elongation, because termination is generally indicative of success. For the fixed-length tasks (Follow and Chase), we default to using positive rewards (min = 0 constraint). Although this is arbitrary, our experience is that positive rewards make for more intuitive interpretation of reward tree structures. We stress that this is anecdotal; the relative interpretability of differently-signed rewards is worthy of deeper investigation.

6.3.2 Tree Growth and Pruning

Recall that all tree growth methods in this thesis make use of some split quality criterion, which is greedily maximised with respect to the leaf to be split $x \in \mathcal{X}$, the feature $d \in \{1, \dots, D\}$ and the threshold $c \in \mathbb{R}$. In the previous chapter, we based our criterion on the variance in the return estimates of trajectories that pass through each leaf (see Equation 5.8), thereby making the reward tree growth stage functionally equivalent to CART. While very fast to compute, this criterion is only loosely aligned with the ultimate objective of reward learning, which is to learn a good predictive model of human preferences. For this reason, it seems intuitive that performance could be improved by adopting the more direct criterion of greedily minimising a preference loss over the dataset, and this indeed turns out to be the case.

Resurrecting a piece of notation from Section 4.3.1, let $\mathcal{X} \rightarrow (x, d, c)$ denote the enlarged tree that would result from adding the split parameterised by x , d and c to an extant tree \mathcal{X} . A split criterion based on direct preference loss reduction can be expressed as follows:

$$(6.2) \quad \operatorname{argmin}_{x \in \mathcal{X}, 1 \leq d \leq D, c \in \mathcal{C}_d} \left[\sum_{(i,j) \in \mathcal{L}} \ell(\Pr(\xi^j \succ \xi^i | R_{\mathcal{X} \rightarrow (x,d,c)}) - \ell(\Pr(\xi^j \succ \xi^i | R_{\mathcal{X}})) \right],$$

where (as per in Equation 5.11) the loss for each preference can be decomposed as

$$\ell(\Pr(\xi^j \succ \xi^i | R_{\mathcal{X}})) = \ell(f(\sum_{x \in \mathcal{X}} \text{reward}(x) \times (|\{\mathbf{f}_t^j \in \xi^j : \mathbf{f}_t^j \in x\}| - |\{\mathbf{f}_t^i \in \xi^i : \mathbf{f}_t^i \in x\}|))).$$

We first tried implementing this criterion with the negative log-likelihood loss $\ell(p) = -\log(p)$ and the Bradley-Terry model $f(z) = 1/(1 + \exp(-z))$, which are used in our improved trajectory return estimation method as well as most prior work, but found this to be very computationally costly and prone to overfitting. Instead, we follow Wirth et al. [266] in adopting a discrete 0-1 loss, which considers only the directions of predicted preferences rather than their strengths:

$$(6.3) \quad \ell_{0-1}(p) = [p < 0.5], \text{ or equivalently, } \ell_{0-1}(f(z)) = [z < 0].$$

Note that we use the subscript to differentiate this loss as a special case, and that the latter equation holds because link functions f are always non-decreasing, with a constraint

$f(z) + f(-z) = 1$, so that $f(0) = 0.5$. As a result, the ℓ_{0-1} loss is invariant to the choice of f , so we can bypass using one entirely (which reduces computational overhead). Instead, we can write the loss for a pairwise trajectory preference $\xi^j \succ \xi^i$ as follows:

$$(6.4) \quad \ell_{0-1}(\Pr(\xi^j \succ \xi^i | R_{\mathcal{X}})) = [(\sum_{x \in \mathcal{X}} \text{returndiff}(x, i, j)) < 0],$$

where $\text{returndiff}(x, i, j) = \text{reward}(x) \times (|\{\mathbf{f}_t^j \in \xi^j : \mathbf{f}_t^j \in x\}| - |\{\mathbf{f}_t^i \in \xi^i : \mathbf{f}_t^i \in x\}|)$ is shorthand for the difference in return that ξ^i and ξ^j obtain from leaf x . If the sum of these differences across the tree is negative, the loss for the preference is 1. Otherwise, the loss is 0. When evaluating the splitting of a leaf x into two new leaves $x^{(d \geq c)}$ and $x^{(d < c)}$, the ℓ_{0-1} loss of the enlarged tree $\mathcal{X} \rightarrow (x, d, c)$ for this preference is

$$(6.5) \quad \begin{aligned} \ell_{0-1}(\Pr(\xi^j \succ \xi^i | R_{\mathcal{X} \rightarrow (x, d, c)})) &= [(\sum_{x' \in \mathcal{X} \rightarrow (x, d, c)} \text{returndiff}(x', i, j)) < 0] \\ &= [(\text{returndiff}(x^{(d \geq c)}, i, j) + \text{returndiff}(x^{(d < c)}, i, j)) < -(\sum_{x' \in \mathcal{X} \setminus \{x\}} \text{returndiff}(x', i, j))]. \end{aligned}$$

Therefore, the split can flip the loss from 1 to 0 in cases when the (positive) returndiff across the two new leaves exceeds the (negative) returndiff from the rest of the tree. By Equation 6.2, the split that will be chosen during greedy tree growth is the one that maximises the number of these $1 \rightarrow 0$ flips, minus the number of undesirable $0 \rightarrow 1$ flips, across all preferences in the dataset. Ties between multiple greedy-optimal splits are broken uniform-randomly.

We have developed a parallelised, just-in-time compiled implementation of this split evaluation calculation, thereby making it usable in practice. Although it remains more computationally costly than the variance-based criterion, it is nonetheless a moderate part of the overall cost of online reward learning. As in the previous chapter, we grow reward trees by recursive splitting until either no loss reduction can be achieved by any single split or a tree size limit $|\mathcal{X}| = M_{\max}$ is reached. We then move to a pruning stage, which is identical to that outlined in Section 5.3.4, except that we continue to use the ℓ_{0-1} loss instead of the least squares loss. In Section 6.4, we show that switching to using ℓ_{0-1} as the basis for tree growth and pruning consistency improves the performance of reward trees and RL agents trained using them.

6.3.3 Model-based RL Agents

Online reward learning methods are generally agnostic to the algorithm used for policy learning, and this modularity is hailed as an advantage over other human-agent feedback paradigms [155]. In line with most recent work [44, 154], our experiments in the previous chapter use a model-free RL algorithm, specifically soft actor-critic (SAC) [107]. SAC is widely seen as a strong baseline RL technique for continuous control, and is effective at ensuring exploratory policies due to its explicit entropy maximisation. However, some reward learning methods instead use model-based RL agents that leverage learnt dynamics models and planning [165, 204, 207].

Model-based RL is attractive in the reward learning context because it separates the predictive and normative aspects of decision-making. Since (assuming no changes to the

environment) dynamics remain stationary during online reward learning, the amount of re-learning required is reduced and along with it, the risk of pitfalls such as manipulation [12] and premature convergence. Additionally, given appropriate pre-training, model-based methods are often able to learn from far fewer environment interactions than model-free alternatives. This is especially important in contexts with a human-in-the-loop, whose time is effectively wasted if they need to wait for long periods of ‘dead time’ between successive preference queries. The dynamics of the aircraft handling environment are generally smooth, in that the state (aircraft positions, velocities, orientations, etc.) evolves gradually as a continuous function of the previous state and action. This property makes it easier to learn a good dynamics model.

For these reasons, the experiments in this chapter use a model-based RL algorithm called PETS [45]. PETS selects actions by decision-time planning through a learnt dynamics model $T' : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ up to a horizon H_p . At a given point in online reward learning, let \mathcal{X}^* denote the current reward tree after pruning. With the agent in state $s \in \mathcal{S}$, planning searches for a sequence of H_p future actions that maximise expected return under \mathcal{X}^* :

$$(6.6) \quad \operatorname{argmax}_{(a_0, \dots, a_{H_p-1}) \in \mathcal{A}^{H_p}} \mathbb{E}_{T'} \left[\sum_{t=1}^{H_p} \gamma^{t-1} R_{\mathcal{X}^*}(\phi(s_{t-1}, a_{t-1}, s_t)) \right], \text{ where } s_0 = s, s_{t+1} \sim T'(\cdot | s_t, a_t),$$

$\gamma \in [0, 1]$ is a discount factor, and ϕ is the feature function that maps (state, action, next state) transitions into D -dimensional feature vectors. The first action $a = a_0$ is executed, and then the agent re-plans on the next timestep. In practice, T' is an ensemble of probabilistic neural networks trained to minimise prediction error on an exploratory dataset, and the expectation over T' is replaced by a sample estimate. The optimisation is approximated by the iterative cross-entropy method, whereby candidate action sequences are sampled from an independent Gaussian whose parameters are updated towards the highest-return sequences from the previous iteration. For more details on the PETS algorithm, we refer the reader to the original paper [45].

In the aircraft handling domain, we find that switching from SAC to PETS reduces environment interaction during reward learning by orders of magnitude, and cuts wall-clock runtime, for the same asymptotic performance. A demonstration of this trend on the Follow task is given in Section 6.4.6. In addition, the switch from SAC to PETS has implications for how trajectory pairs can be sampled for preference labelling, as described below.

6.3.4 Online Learning Setup

With a model-free RL algorithm such as SAC, policy learning is gradual. It depends on gradient-based updates to both a policy network and a separate value network, as well as auxiliary target networks which are updated in a lagging way. As a result, when a reward tree that a SAC agent uses as its objective is updated on a new batch of preferences, it takes some time for that change to ‘filter through’ to the policy itself. In contrast, model-based algorithms such as PETS plan afresh on every timestep, so are able to immediately exploit the latest reward tree as soon as it is updated. This means that each online trajectory generated by a PETS

agent can be assumed to be near-optimal¹ for the current tree.

This difference permits a simplified method of trajectory sampling for querying human (or oracle) preferences. Suppose that the PETS agent has just generated the i th trajectory, ξ^i , which is appended to the existing set Ξ . In place of the optimistic sampling scheme described in Section 5.4.3 (which adds computational overhead by re-evaluating the return of each trajectory in Ξ after every tree update), we simply ask the human/oracle to compare ξ^i itself to K_{batch} uniform-randomly sampled trajectories from Ξ . Because ξ^i will be near-optimal for the current reward tree, this approach of always including it in every preference query acts as a simple form of optimistic sampling to correct reward overestimation. This method is also justified by recent analysis by Hu et al. [120], which finds that *near on-policy* preference queries using the latest agent behaviour leads to reward models that are maximally useful for continued policy learning.

As a further simplification, we find that scheduling preference collection into increasing-sized batches (as per Section 5.4.4) no longer confers a benefit when learning with PETS agents in the aircraft handling environment. Instead, we use uniform batch sizes. Since, as discussed, PETS agent trajectories immediately optimise for the latest reward tree with no lag, the benefit of obtaining preferences early outweighs the benefit of deferring them until some initial policy improvement has occurred. In the notation of the previous chapter, our simplified method collects a fixed $K_{\text{batch}} = K_{\text{max}}/N_{\text{max}}$ preferences every time a single new trajectory is generated (i.e. $N_{\text{batch}} = 1$), then immediately updates the reward tree (i.e. $K_{\text{tree}} = K_{\text{batch}}$).

In a final change to the online learning setup, we find that our original method of resuming growth from the current state of the tree in each successive update causes lock-in to poor initial split selections in the aircraft handling environment. Instead, we re-grow a reward tree from scratch on each update. The rule structure nonetheless tends to stabilise as the enlarging preference dataset becomes increasingly similar for later updates. Early experiments indicated that the lock-in problem is mitigated by this change, resulting in more sustained improvements in preference loss and agent performance. Since more splits are evaluated and made per update step, computation time is increased. However, when typical post-pruning tree size $|\mathcal{X}^*|$ (≈ 20 in our experiments) is small compared with M_{max} ($= 100$), this increase is marginal, and contributes only a few percentage points to overall runtime.

We did not encounter lock-in in the simpler environments of the previous chapter, and suspect that the greater diversity of possible behaviour in large state spaces creates more risk of causal confusion and very misaligned early splits. Since the growth resumption method is faster (and enables the kind of traceability explored in Figure 5.12), but from-scratch growth may improve performance, the best choice may be case-specific. We speculate that something akin to the simultaneous growth and pruning strategy developed for the CSTA algorithm in Section 4.7 could improve resumption performance, although we are yet to verify this.

¹The degree of optimality depends on the error in the learnt dynamics model T' (which is very low in our aircraft handling experiments), as well how successfully the iterative planning algorithm converges. Planning for more iterations, and with more candidate action sequences per iteration, enables a closer approach to optimality.

6.4 Experiments and Results

In this section, we combine quantitative and qualitative evaluations to assess the performance of our refined reward tree learning method on the aircraft handling tasks, specifically in comparison to the standard approach of using neural networks (henceforth, NNs). We also assess the benefit of switching from variance-based to ℓ_{0-1} -based split selection during tree growth, and from model-free to model-based RL agents. In place of costly human-in-the-loop evaluation with real aviation experts, all experiments use synthetic oracle preferences with respect to the ground truth reward functions given in Section 6.2.2. However, the sensitivity analysis in Section 6.4.5 aims to improve the realism of the evaluation by examining the performance impacts of noisy and myopic oracles and a restricted data budget.

6.4.1 Common Parameters

Tree learning parameters In all cases, trajectory return estimation uses the Adam optimiser with a learning rate of 0.1, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Optimisation stops when the mean negative log-likelihood changes by less than $1e^{-5}$ between successive gradient steps. As mentioned in Section 6.3.1, we enforce negative rewards (max = 0 constraint) for the Land task, and positive rewards (min = 0 constraint) for Follow and Chase. We use a maximum tree size of $M_{\max} = 100$ to terminate tree growth and a regularisation parameter of $\alpha = 0.005$ during pruning.

PETS RL algorithm We learn a dynamics model T' using an ensemble of five feedforward neural networks, each with four hidden layers of 200 hidden units and ReLU activations. State vectors are pre-normalised by applying a hand-specified scale factor to each dimension. Decision-time planning operates over a time horizon of $H_p = 10$ and consists of 10 iterations of the cross-entropy method. Each iteration samples 20 candidate action sequences from an independent Gaussian, of which the top five in terms of return are identified as ‘elites’, then updates the sampling Gaussian towards the elites with a learning rate of 0.5. In all experiments, we use $\gamma = 1$, meaning no temporal discounting is applied during planning.

Dynamics pre-training In our experiments, we find that the particular dynamics of the aircraft handling environment permit us to pre-train T' on random offline data, and accurately generalise to states encountered during online reward learning. This means we perform no further updates to the model while reward learning is ongoing. As well as improving wall-clock speed, this avoids complexity and convergence issues arising from having two interacting learning processes (note that simultaneous learning is completely unavoidable with model-free RL). To pre-train, we collect $1e^5$ transitions by rolling out a uniform random policy, then update each of the five networks on $1e^5$ independently sampled mini-batches of 256 transitions, using the mean squared error loss over normalised next-state predictions.

Neural network reward learning baseline We baseline our reward tree models against the de facto standard approach of reward learning using a NN, with preference losses computed

as negative log-likelihoods under the Bradley-Terry model. In constructing this baseline, we retain as much of the overall learning pipeline as possible, so that only the model architecture varies. The result is that we replace the four-stage tree update process with the following:

```

1: for  $b \in \{1, \dots, B\}$  do
2:    $\mathcal{L}_{\text{mini-batch}}$  = a mini-batch of  $K_{\text{mini-batch}}$  preference labels sampled uniformly from  $\mathcal{L}$ 
3:   Initialise loss sum = 0
4:   for  $(i, j) \in \mathcal{L}_{\text{mini-batch}}$  do
5:     Predict trajectory returns  $g_i = \sum_{t=1}^{H^i} R_{\text{NN}}(\mathbf{f}_t^i)$  and  $g_j = \sum_{t=1}^{H^j} R_{\text{NN}}(\mathbf{f}_t^j)$ 
6:     Compute negative log-likelihood  $-\log(1/(1 + \exp(g_i - g_j)))$  and add to loss sum
7:   end for
8:   Backpropagate gradient of loss sum through  $R_{\text{NN}}$  and update parameters
9: end for
10:  $\mathbf{r}_{\text{all}}$  = reward predictions  $R_{\text{NN}}(\mathbf{f})$  for all feature vectors  $\mathbf{f}$  in the trajectory set  $\Xi$ 
11: Scale network outputs by  $1/\text{std}(\mathbf{r}_{\text{all}})$ 
12: Shift network outputs by  $-\min(\mathbf{r}_{\text{all}})$  or  $-\max(\mathbf{r}_{\text{all}})$ , depending on desired reward sign

```

$R_{\text{NN}} : \mathbb{R}^D \rightarrow \mathbb{R}$ denotes the NN reward model, and lines 10-12 replicate the standard deviation and sign normalisation applied in Section 6.3.1. Preference collection is performed identically and the same PETS agent implementation is used, except that rewards are provided by R_{NN} instead of a tree. We follow the popular PEBBLE baseline [154] in implementing R_{NN} as a three-layer network with 256 hidden units each and leaky ReLU activations, and performing the update on line 8 using the Adam optimiser with a learning rate of $3e^{-4}$. On each update, we sample $B = 100$ mini-batches of size $K_{\text{mini-batch}} = 32$ and take one gradient step per mini-batch.

6.4.2 Online Performance Evaluation

In our main experiments, we use ideal, error-free oracles, which always prefer trajectories with higher ground truth returns. We evaluate online reward learning using trees with the ℓ_{0-1} split criterion, comparing to the previous chapter’s variance-based criterion, as well as the NN baseline. For each of the three aircraft handling tasks, we collect $K_{\text{max}} = 1000$ preferences over $N_{\text{max}} = 200$ online trajectories generated by a PETS agent, and run 10 repeats.

As a headline statistic, we define the *oracle regret ratio* (ORR) as the median drop in ground truth oracle return of PETS agents deployed for 100 episodes using each fully-trained reward model compared with directly using the ground truth reward, taken as a fraction of the drop to a uniform random policy (lower is better). This gives a normalised measure of the alignment of each reward model with the ground truth, which is comparable across tasks with different intrinsic difficulty levels. Below are the median (top) and minimum (bottom) ORR values across the 10 repeats for each task-model pairing:

Follow			Chase			Land		
NN	Tree (0-1)	Tree (var)	NN	Tree (0-1)	Tree (var)	NN	Tree (0-1)	Tree (var)
0.000	0.120	0.284	-0.030	0.040	0.126	0.014	0.050	0.062
-0.010	0.057	0.158	-0.051	-0.011	0.065	-0.030	0.011	0.010

In these results, we observe that:

- All models yield agent performance that is far closer to the oracle ($\text{ORR} = 0$) than random ($\text{ORR} = 1$), in most cases just a few percentage points below the oracle;
- NNs outperform trees by a small but variable amount;
- ℓ_{0-1} splitting consistently outperforms the variance-based method;
- both NN and tree models sometimes exceed the direct use of the oracle (negative ORR).

The latter, counter-intuitive, phenomenon has been observed elsewhere in the reward learning literature [37, 80, 202]. It may be due to the learnt reward being better *shaped* than the ground truth [188], making it easier for the PETS planner to converge to near-optimal action sequences.

Figure 6.3 expands these results with more metrics, revealing temporal learning trends not captured by headline ORR values. Metrics are plotted as time series over the 200 learning episodes (sliding-window medians and interquartile ranges (IQRs) across repeats). In the top row (a), the ORR of online trajectories shows how agent performance converges over time. For Follow, there is a gap between the models, with ℓ_{0-1} splitting visibly aiding performance over variance-based splitting, and potentially enabling somewhat faster early learning than the NNs, although it converges to a higher asymptote. The learning curves for Chase and Land are much more homogeneous, and the NNs reach only slightly lower asymptotes, with overlapping IQRs. The majority of runs converge to their final performance well within the 200 episode learning period; this learning speed is made possible by the use of model-based PETS agents.

The second row of plots (b) shows how the discrete preference loss ℓ_{0-1} of each reward model changes over time. The loss tends to increase as the growing preference dataset presents a harder prediction problem, though the shapes of all curves suggest convergence. Random preference prediction gives $\ell_{0-1} = 0.5$ in expectation, and all models remain far below this value even on the largest (final) preference datasets. For Follow and Land, the trees that directly split on ℓ_{0-1} actually achieve lower loss than the NNs; they more accurately predict the direction of preferences in the dataset. This is an encouraging result, indicating that our models perform well on the metric for which they are directly optimised, but the fact that this does not translate into lower ORR indicates that the problems of learning a good policy and replicating a given preference dataset are not perfectly correlated. This subtle point has been made before [159].

A potentially important factor in these experiments is that the oracle reward for Follow is a linear function, while the others contain progressively more terms and discontinuities (see Section 6.2.2). A trend suggested by these results is thus that the performance gap between NNs and reward trees (on both ORR and ℓ_{0-1} metrics) reduces as the ground truth reward becomes more complex and nonlinear. Although further experiments would be needed to test this hypothesis thoroughly, it is backed up by results on other metrics in the next subsection.

For the reward tree models, the bottom row of plots (c) shows how the number of leaves changes over time. There is notable consistency in these trends between the repeated runs.

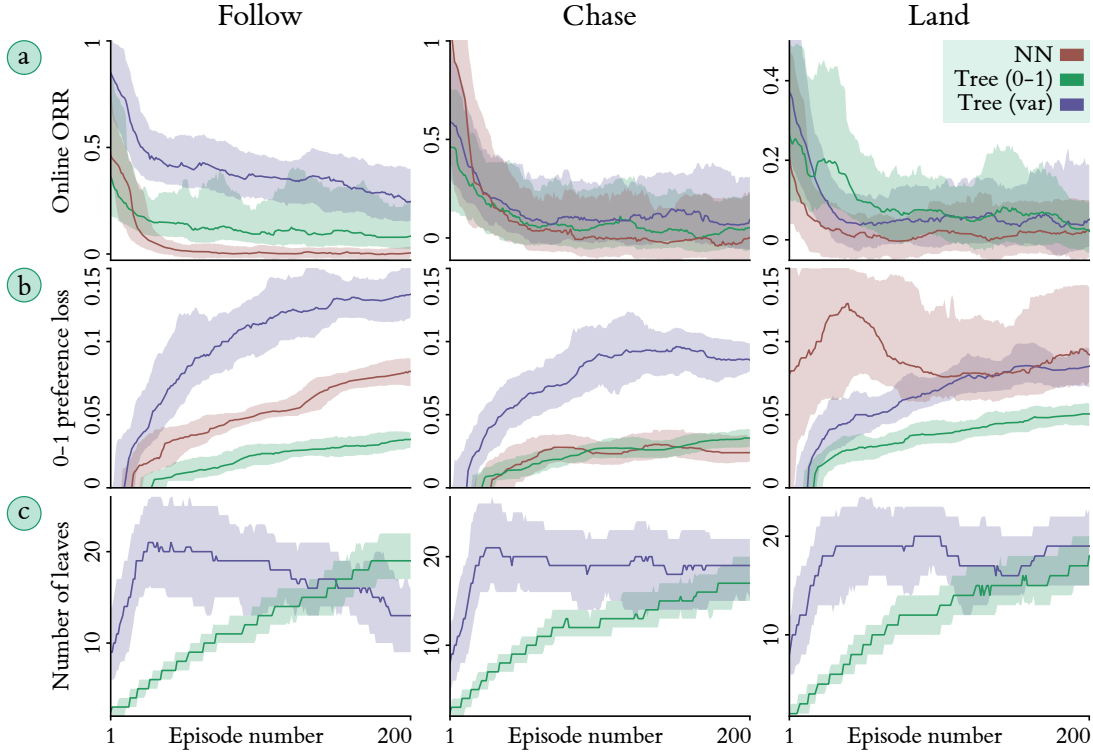


Figure 6.3: Time series of metrics for online NN- and tree-based reward learning.

While the variance-based trees tend to grow rapidly initially before stabilising or shrinking (echoing the trend identified in the previous chapter), the ℓ_{0-1} trees enlarge more conservatively, suggesting this method is less liable to overfit to small preference datasets. This may be a major factor in the improved performance of the new split criterion. Trees of a readily interpretable size (≈ 20 leaves) are produced for all tasks. While this is encouraging, it is possible that performance could be improved by independently tuning the size regulariser α per task.

6.4.3 Policy-Invariant Evaluation

Gleave et al. [99] recently highlighted the importance of comparing and evaluating learnt reward models in a policy-invariant manner, by using a common evaluation dataset rather than on-policy data generated by agents optimising for each model. This enables a truly like-for-like comparison, unbiased by the particular distribution of states that each model incentivises. Ideally, the offline evaluation data should have high coverage (i.e. high-entropy state distribution, both high- and low-quality trajectories), in order to characterise the reward models' outputs across a spectrum of plausible policies. We report such a comparison in Figure 6.4, for which the evaluation datasets are generated by PETS agents using the oracle reward functions, with added action randomisation to increase coverage.²

²Specifically, we deploy PETS using the oracle reward, but randomise its hyperparameters (number of planning iterations $\in \{1, \dots, 50\}$, number of action sequence samples $\in \{4, \dots, 50\}$) on every episode. In all

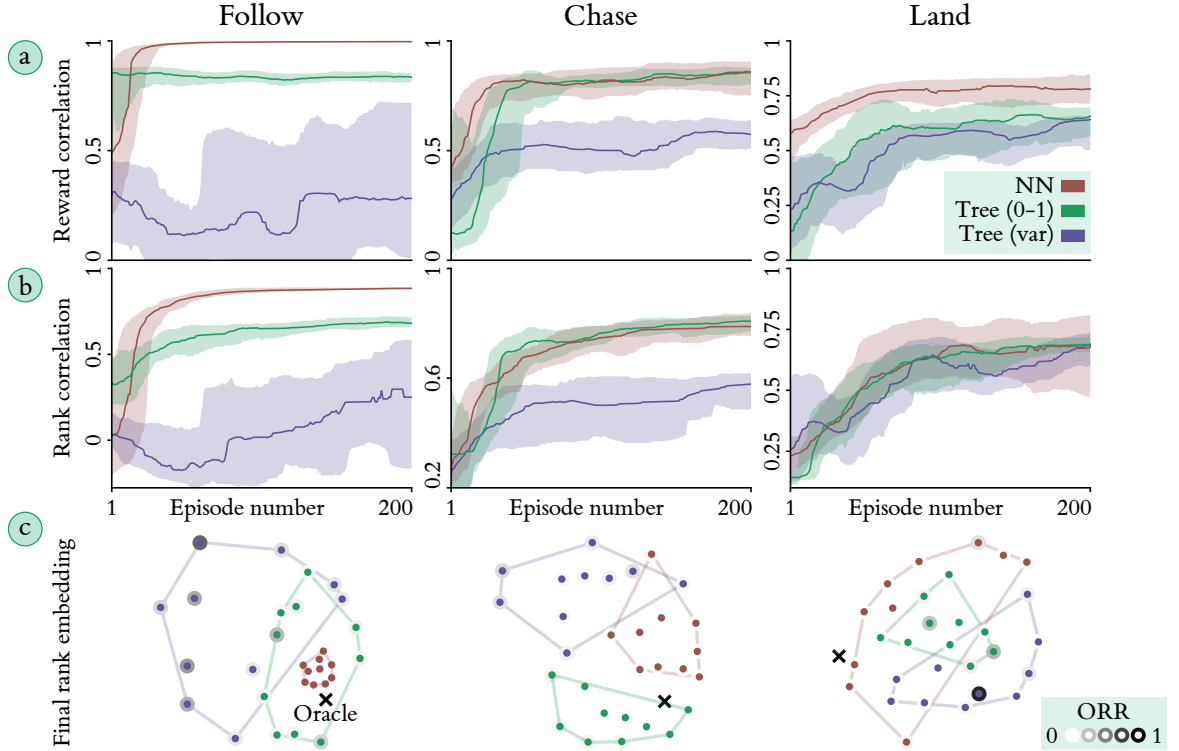



Figure 6.4: Policy-invariant evaluation of online NN- and tree-based reward learning.

In the top row (a), we measure the Pearson correlation of each model’s reward predictions on the policy-invariant dataset with those of its respective oracle at each point in learning. This indicates each model’s ability to generalise to unseen transitions that are not included in the preference dataset. For Follow, the NN models achieve near-perfect correlation, but more interestingly, the gap between variance-based and ℓ_{0-1} -based splitting is very large. This indicates that our novel split criterion can create reward trees that generalise far more effectively to unseen predictions (with far lower variance across repeated runs). ℓ_{0-1} splitting maintains a strong advantage on the Chase task, and in fact matches the NN models. For Land, the gap between all three models is smallest, and the benefit of ℓ_{0-1} splitting is less conclusive.

In the middle row (b), we use each reward model to predict the return of each trajectory in the policy-invariant dataset, rank these returns from lowest to highest, then relate this to the true ranking according to the oracle using Kendall’s τ rank correlation coefficient [139]. Because this measure is defined in terms of pairwise comparisons, it indicates the effectiveness of each model for predicting randomly sampled oracle preferences over the (unseen) policy-invariant trajectories. The curves subtly differ from those in (a), indicating that it is possible to reconstruct trajectory preferences to a given accuracy with varying fidelity at the individual reward level.

cases, we take the top 25% of action sequences as elites. This randomisation results in trajectories that are sometimes near-optimal with respect to the oracle, sometimes moderate in quality, and sometimes barely better than random. For all three tasks, we generate a dataset of 200 evaluation trajectories in this manner.

However, the common overall trend persists: ℓ_{0-1} -based trees outperform variance-based ones, with NNs sometimes improving again by a smaller margin, and sometimes bringing no added benefit. Moving left-to-right across the tasks, the gap between models reduces from both sides; NN performance worsens while variance-based trees improve. This reinforces the suggestion in the preceding subsection that the NN-tree gap narrows for more complex and nonlinear tasks.

The bottom row  presents a novel form of reward visualisation. Given some measure of similarity between reward functions, such as those considered in the first two rows, we can compute a matrix of pairwise similarities between any number of such functions (computational cost permitting). We can then produce a 2D embedding of the functions by applying multidimensional scaling (MDS). Visualising this embedding as a scatter plot enables the discovery of salient patterns and trends in the set of functions. In the plots shown, we use trajectory rank correlation on the policy-invariant datasets as the similarity measure, and the SMACOF MDS algorithm [62], to embed all ($3 \times 10 =$) 30 model repeats and the oracle for each task. This gives an impression of the models’ similarity not just to the oracle, but to each other.

Aside from the Follow NNs, which form a tight cluster near the oracle (providing further evidence that it has been reconstructed near-perfectly), the distribution for each model indicates roughly equal consistency between repeats. This suggests that NNs and reward trees have similar levels of robustness to changes in the precise content of the trajectory preference dataset. The overlap of convex hulls suggests that the rankings produced by all models are broadly similar for Land, but more distinct for Chase. Shading points by ORR reveals that while models further from the oracle tend to induce worse-performing policies, the trend is not monotonic. This reinforces the point made above that the problems of learning a good policy and exactly replicating the ground truth reward are not identical.

Populating such embedding plots more densely, perhaps by varying model hyperparameters, could provide a means of mapping the space of learnable reward functions and its relationship to policy performance. It would also be straightforward to compute similarity values for the same model repeat at multiple checkpoints during learning. This would yield a trajectory in the 2D embedding space, which could aid the assessment of the stability and convergence properties of online learning with different models and hyperparameter values.

6.4.4 Visual Trajectory Inspection

While useful for benchmarking, quantitative metrics provide little insight into the behaviours incentivised by each reward model. They would also mostly be undefined when learning from real humans since the ground truth reward is unknown. We therefore complement them with a visual analysis of induced agent behaviour. Figure 6.5 plots 500 trajectories of PETS agents using the best repeat by ORR for each task-model combination, across a range of features as well as time (see Table 6.1 for a reminder of feature definitions). Each trajectory is coloured on a red-blue scale according to its ORR. Dashed black curves indicate the single trajectory

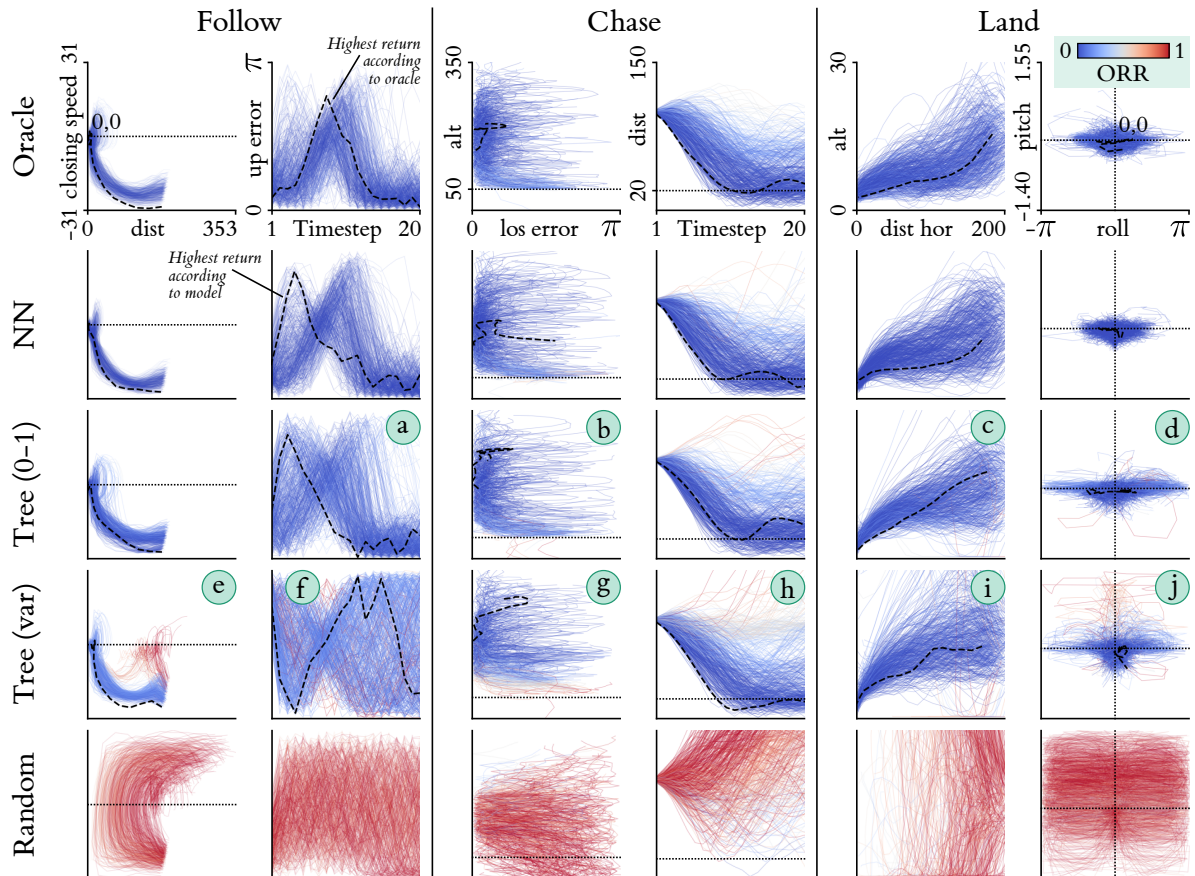


Figure 6.5: Agent trajectories using the best models by ORR (oracle and random for comparison).

with the highest predicted return according to each model. We also show trajectories for PETS agents with direct oracle access, which serve as the benchmark behaviour that we aim to match via reward learning, and for random policies, which perform very poorly on all three tasks.

In general, all models are far closer to the oracles than random, with few examples of obviously unstable handling behaviour or task failure (highlighted in red, due to colouring by ORR). While the NNs induce trajectories that are almost indistinguishable from the oracles, ℓ_{0-1} -based reward trees lag not far behind. Variance-based trees create more anomalies.

Successes of the ℓ_{0-1} trees include the execution of Follow with a single banked turn before straightening up, as shown by the **up error** time series **(a)**, where **up error** = 0 is level flight. Interestingly, both this tree and the NN reward model appear to favour a somewhat earlier turn than the oracle (i.e. peak shifted to the left on this plot). Indeed, the trajectories for the ℓ_{0-1} tree are almost imperceptibly different from those of the NN, despite their quantitative performance (e.g. ORR) differing. This underlines the importance of joint quantitative-qualitative evaluation. **(b)** shows that for Chase, the ℓ_{0-1} tree has clearly learnt the most safety-critical aspect of the task, which is to keep the agent above the altitude threshold **alt** < 50, below which the oracle reward is strongly negative. The threshold is violated in only eight of 500 trajectories

(1.6%). Further evidence that this altitude threshold has been learnt correctly is presented in Section 6.5. For Land, the ℓ_{0-1} tree replicates the oracle in producing a gradual reduction in `alt` (c) while usually keeping `pitch` close to 0 (d). However, the spread of `roll` values is somewhat wider, suggesting that the tree is insufficiently penalising deviations from level flight.

In marked contrast to the above, the agent using the variance-based tree for Follow sometimes fails to reach the target position (e; red trajectories), and also does not reliably straighten up to reduce up `error` (f). For Chase, the altitude threshold does not appear to have been learnt precisely, and lower-altitude trajectories often fail to close the distance to RJ (g and h; red trajectories). For Land, the variance-based tree gives a later and less smooth descent (i), and less consistent pitch control (j), than the NN or ℓ_{0-1} -based tree, although all models produce a somewhat higher altitude profile than the oracle. Overall, it is clear that the benefit of switching from variance-based to ℓ_{0-1} -based split selection manifests not only in quantitative metrics, but also in qualitative behavioural improvements.

6.4.5 Sensitivity Analysis

It is important to consider how performance degrades with reduced or corrupted data. We thus evaluate the effect of varying the number of preferences K_{\max} (with fixed $N_{\max} = 200$) and trajectories N_{\max} (with fixed $K_{\max} = 1000$) on reward learning with NNs and ℓ_{0-1} -based trees. We also create more human-like preference data via two modes of oracle “*irrationality*” proposed by Lee et al. [153]. The first is noise, which we introduce by generating oracle preferences using variants of the Bradley-Terry model that are calibrated to give a desired error rate on the policy-invariant datasets.³ This adheres to an assumption of classic human preference models, namely that preference mistakes are more likely for trajectory pairs whose returns are more similar (see Section 5.2). The second is a myopic recency bias, whereby the oracles exponentially discount earlier timesteps (by a factor $\gamma < 1$) when computing trajectory returns. There is evidence that humans commonly exhibit such a bias when retrospectively evaluating episodic experiences [88]. We run five repeats for all cases, and report the medians and IQRs of two performance metrics: the ORR of PETS agents deployed using the final reward models (lower is better) and trajectory rank correlation on the policy-invariant datasets (closer to 1 is better).

As Figure 6.6 shows, both NN and tree models exhibit good robustness with respect to all four varied parameters. Although NNs remain superior in most cases, the gap varies, and is often reduced compared to the base cases (bold labels). Another general observation is that the trends for trees are somewhat smoother than for NNs, with fewer sharp jumps and fewer instances of very high spread across the five repeats. This potentially indicates that reward trees behave in a more consistent way under non-ideal learning conditions.

³In particular, for two trajectories ξ^i and ξ^j , a noisy Bradley-Terry oracle for ground truth reward R prefers the latter with probability $1/(1 + \exp(\beta(G(\xi^i|R) - G(\xi^j|R))))$ for some $\beta > 0$. For each task, β is calibrated by gradient descent to achieve each of the desired error rates (0.1, 0.2, 0.3, 0.4, 0.45) across 1000 randomly sampled preferences over the respective policy-invariant trajectories. Higher error rates require higher values of β .

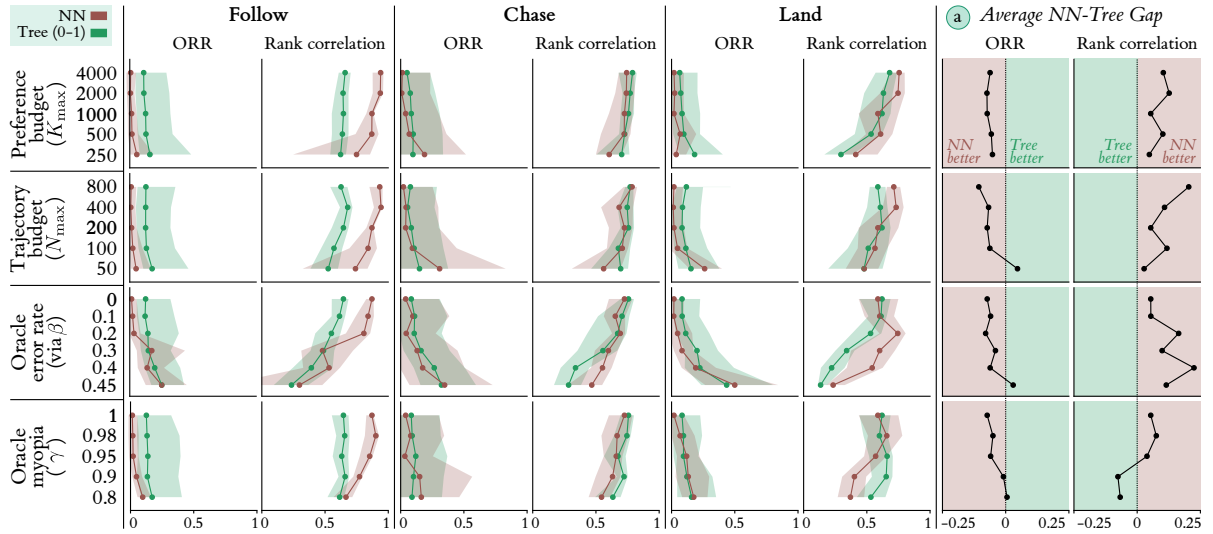


Figure 6.6: Comparative sensitivity analysis of reward learning with NNs and trees.

The sensitivity to preference and trajectory budgets is low, with little improvement for $K_{\max} > 1000$ and $N_{\max} > 200$, and no catastrophic drop even with 25% of the data as the base case. In several contexts (K_{\max} study for Follow and Chase, N_{\max} study for Chase and Land) the performance dropoff on both metrics seems to be somewhat more gradual for reward trees than for NNs, to the extent that trees sometimes perform better under the most restricted budgets. This suggests that reward trees can make effective use of small preference datasets, and may be less liable to overfit to them than other model classes.

For all tasks, the oracle error rate can increase to around 20% before large drops in performance are observed. Trees in particular remain virtually unaffected by an oracle myopia factor as low as $\gamma = 0.9$, but exhibit more of a dropoff once this reaches 0.8. The Land task appears to be somewhat more sensitive than the others to both modes of oracle irrationality.

In the right column, labelled as **a**, we summarise these results by taking the difference between the NN and tree metrics, and averaging across the three tasks. In all cases aside from rank correlation with error-prone oracles, the NN-tree gap tends to become more favourable to the trees as the varied parameter becomes more challenging (top-to-bottom). This sensitivity analysis thus indicates that reward trees are at least as robust to difficult learning scenarios as NNs, and may even be slightly more so. This is a promising result for the transferability of reward tree learning to limited and imperfect datasets of real human preferences.

6.4.6 Comparison to Model-free Reinforcement Learning

This subsection justifies the switch to model-based PETS agents for the experiments in this chapter, through direct comparison to model-free SAC agents. To summarise, a consistently observed benefit of model-based RL is its sample efficiency, and we find that this trend holds in our context. Running our reward learning pipeline unchanged except for the use of SAC agents

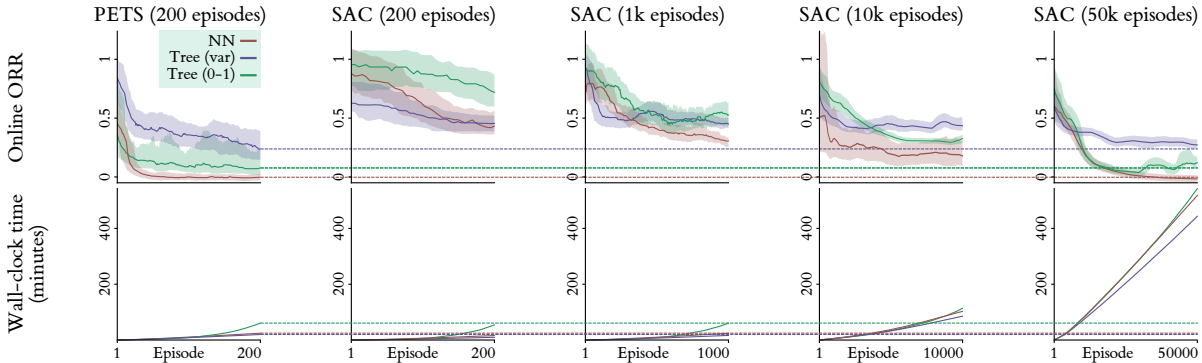


Figure 6.7: Comparing the use of PETS and SAC agents on the Follow task.

for policy learning, we find that over two orders of magnitude more environment interaction is required to achieve equivalent performance in terms of ORR. In turn, this increases runtime by 10-20 \times , thereby outweighing the higher per-timestep computational cost of PETS over SAC.

Figure 6.7 shows these results in detail. The PETS results are taken directly from Figure 6.3. For SAC we retain the default preference budget of $K_{\max} = 1000$, but for longer runs add episode trajectories to Ξ at a reduced frequency so that $N_{\max} = 200$ (e.g. for 50000 episodes, only 1 in 250 episodes are added to the dataset; the rest are skipped). All SAC agents use policy and value networks with two 256-unit hidden layers each, learning rates of $1e^{-4}$ and $1e^{-3}$ for the policy and value updates with the Adam optimiser, a discount factor of $\gamma = 0.99$, and an interpolation factor of 0.99 for Polyak averaging of the target networks. Updates use mini-batches of 32 transitions sampled uniformly from a rolling replay buffer of capacity $5e^4$.

Initially running SAC for a total of 200 episodes, matching our PETS experiments, gives the model-free learning algorithm insufficient time to achieve good performance in terms of ORR (a higher learning rate leads SAC to become unstable). We then progressively increase the length of SAC runs until performance matches the use of PETS, and find that this requires around 50000 episodes, an increase of 250 times.⁴ In terms of wall-clock runtime (on a single NVIDIA Tesla P100 GPU), running reward learning with SAC for 1000 episodes is roughly equivalent to 200 episodes using PETS (25-60 minutes, depending on the reward model architecture). For 50000 episodes, this time increases to 9 hours. This brings a practical disadvantage: if reward learning were done using human preferences instead of an oracle, that person would have to dedicate more than a full working day to the exercise, most of which would be ‘dead time’ waiting for several minutes between each successive preference batch.

Note that the PETS wall-clock times quoted here exclude the time to pre-train the dynamics models. Although this is not how sample complexity is typically measured in model-based RL, we argue that it is appropriate for the reward learning context, where the key factor is the period

⁴It is noteworthy that the variance-based tree model seems to perform best in the short-runtime regime but worst in the long-runtime regime. We have not assessed whether this holds in other tasks, but such an investigation would be worthwhile.

for which a human would be required to be in-the-loop. Front-loading the sample complexity and wall-clock time onto the dynamics learning phase (which need not be human-supervised in the same way) further reduces ‘dead time’ as described above. In addition, the same dynamics model may be reusable for multiple runs of reward learning with different human evaluators (assuming no major distributional shift), meaning the effective sample complexity of each run is reduced yet further [165]. Regardless, pre-training in the aircraft handling domain takes around 30 minutes, which remains low compared with the 9 hours for high-performing SAC agents.

6.5 Interpretability Analysis

The central finding of the preceding section is that reward learning with ℓ_{0-1} -based trees can be competitive with NNs on the aircraft handling tasks, but not quite as performant overall. We now return to the motivating advantage of using a reward tree, which may tip practical tradeoffs in its favour: the ability to interpret the learnt structure, analyse how it arises from the underlying dataset of preferences, and trace how it influences agent behaviour. In this section, we present a series of visualisations and analyses, which are complementary to those in Section 5.9 of the preceding chapter. As in similar sections elsewhere in this thesis, we favour depth over breadth, so focus exclusively on the single best tree by ORR on the Chase task.

6.5.1 Tree Structure Appraisal

This reward tree has 17 leaves, as shown in Figure 6.8. This diagram also shows an aggregate reward for each subtree, which is computed via the same process of timestep-weighted averaging as used to assign leaf rewards. This makes it easier to interpret the effect of each split on reward predictions. The ground truth reward used by the oracle, reprinted below for convenience, uses four features, all of which are used in the tree in ways that are broadly aligned (e.g. lower `los error` leads to leaves/subtrees with higher reward). This indicates that the ℓ_{0-1} -based split criterion has been effective at selecting the task-relevant features from the list in Table 6.1.

By inspection, it is clear that the model has learnt the crucial threshold `alt < 50`, correctly assigning low reward when it is crossed (leaves 1 and 7). This explains why we observe rare violations of the altitude threshold in Figure 6.5. However, it has not learnt the ideal distance to the reference jet (RJ), `dist = 20`, with 43.3 being the lowest value used in any of the rules. This could be because the underlying preference dataset lacks sufficient preferences to make this distinction. Running the learning algorithm for longer, or adopting a more sophisticated active querying scheme, may aid the discovery of such subtleties.

Other features besides those used by the oracle are present in the tree, indicating some causal confusion of the kind observed several times in the previous chapter. However, in this case it is less clear that the inclusion of these features is detrimental to agent performance, as they could plausibly provide beneficial shaping (e.g. penalising positive `closing speed`, which

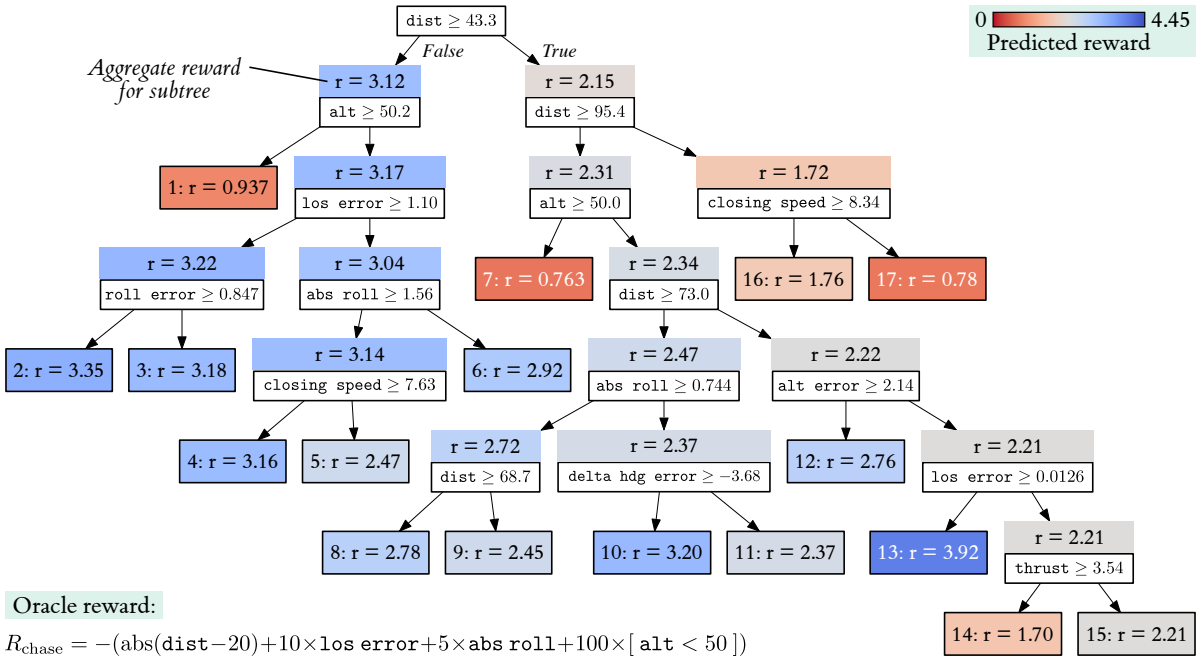


Figure 6.8: Diagram of a reward tree learnt for the Chase task (“r” denotes reward).

indicates increasing distance to RJ). That may indeed be the case for this model, since its headline ORR figure is actually slightly negative.

6.5.2 Leaf-level Alignment

The decomposition of a reward tree into a discrete set of leaves allows us to evaluate alignment at the individual leaf level. Figure 6.9 plots the tree’s predicted reward against the oracle reward for all timesteps in the online PETS-generated trajectory dataset (correlation = 0.903). Each leaf’s predictions lie along a horizontal line. Most leaves, including 1 and 2, can be considered well-aligned on this dataset because their oracle reward distributions are tightly concentrated around low and high averages respectively (note that the absolute scale is irrelevant because rewards can be scaled without affecting behaviour). Leaf 16 has a wider oracle reward distribution with several negative outliers, indicating that it may be a source of misalignment. An optimal tree may split this leaf further, perhaps using the `alt` < 50 threshold.

In general, alignment appears to be better for leaves with higher reward, likely because the online dataset contains relatively few very poor trajectories. This is arguably desirable; a limited capacity tree may be more useful for policy learning if it can distinguish good and very good behaviour, as opposed to poor and very poor. The one anomaly is leaf 13, which contains just a single timestep from ξ^{77} . This trajectory is the eighth best in the dataset by oracle return, but this leaf assigns that credit to a state that seemingly does not merit it, as the distance to RJ is so high (`dist` > 73). This may be an example of suboptimal reward learning, but the fact that its origin can be precisely pinpointed is a testament to the value of interpretability.

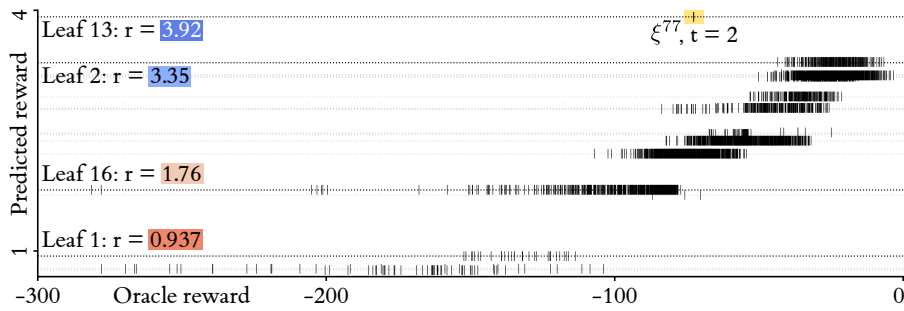


Figure 6.9: Alignment of leaf-level reward predictions with ground truth oracle rewards.

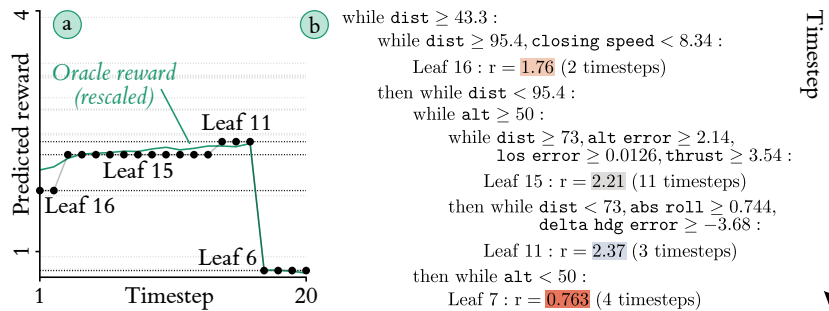


Figure 6.10: Report card explaining the rewards predicted for a trajectory.

6.5.3 Trajectory Report Card

Section 5.9.1 introduced the concept of a reward *report card*, which decomposes the tree-predicted return for a trajectory based on the leaves that are visited, and states the relevant rules. In the aircraft handling context, a report card may be of value to various end users, including a trainee pilot whose own behaviour is being evaluated by the reward tree, and who requires actionable insight into how their score can be improved. Here we present an extension to this method that describes not only which leaves are visited, but *when* the visits occur.

Figure 6.10 considers ξ^{191} from the online trajectory dataset, a rare case that violates the altitude threshold. The time series of reward (a) shows that the 20 timesteps are distributed between leaves 16, 15, 11 and 7. Rescaled oracle rewards are overlaid in teal, and show that the model’s predictions are well-aligned. The report card itself translates this visualisation into a textual form (b), which is similar to a nested program. Read top-to-bottom, and accounting for indentation, the program indicates which rules of the tree are active at each timestep and the effect this has on predicted reward. The first line, and the relative indentation of all others, immediately tells us the entire trajectory is spent with a `dist` of at least 43.3 between the ego and reference jets. This trajectory starts fairly positively, with reward gradually increasing over the first 16 timesteps as `dist` is reduced to between 43.3 and 73, but then falls dramatically when the `alt` < 50 threshold is crossed for the final four timesteps.

Note that this report card conveys equivalent information to the temporal explanatory stories from Section 3.5.4 but emphasises the commonalities between successive timesteps rather than the changes. The relative usefulness of the two techniques is likely to be context-specific.

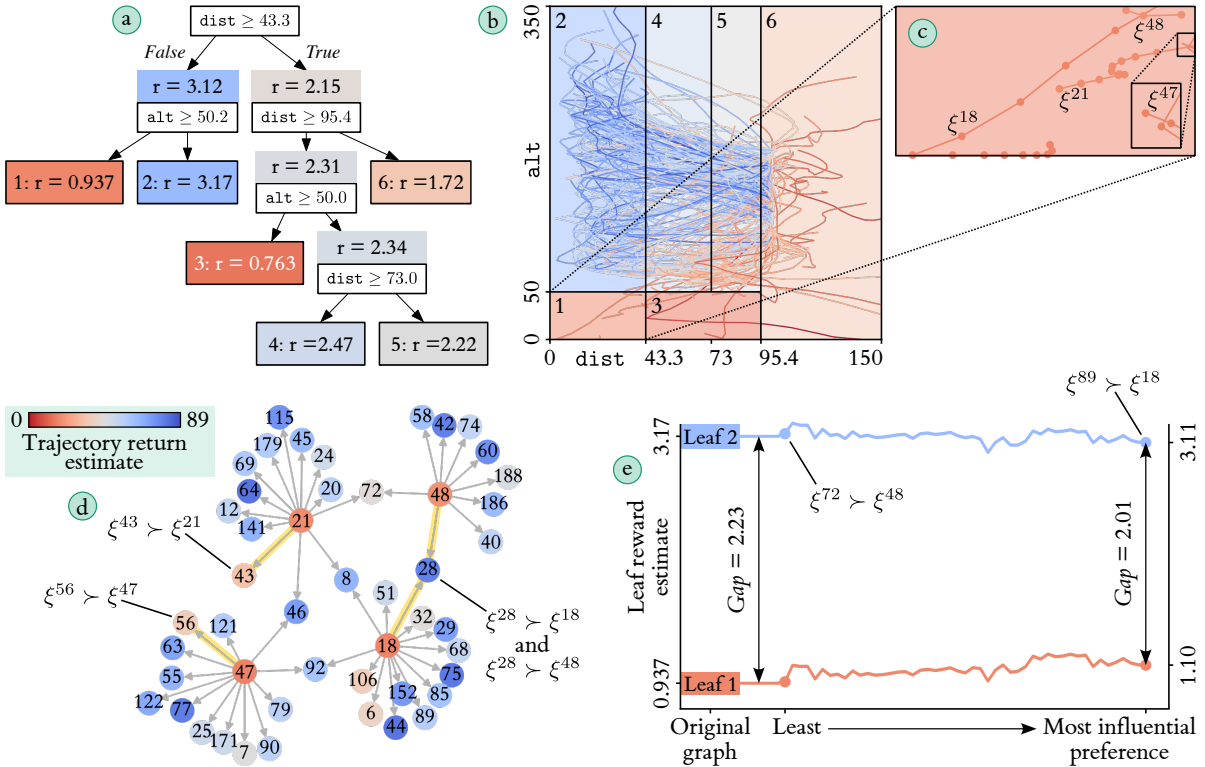


Figure 6.11: Explaining differences in leaf reward predictions by preference reversal.

6.5.4 Preference-based Reward Explanation

We now demonstrate how our entire preference-to-reward learning pipeline, not only the final structure of a reward tree, can be viewed as interpretable, because it provides a mechanism for attributing reward predictions back to individual preferences in the dataset. For the sake of didactic simplicity, we focus on a pruned subtree of the full reward tree, shown in Figure 6.11 (a). Because this subtree splits only on `dist` and `alt`, we can directly visualise its induced feature space partition (b). The 200 online trajectories are overlaid, coloured by their return estimates. Zooming into leaf 1, which covers cases where the altitude threshold is violated, (c) shows that it contains a total of 30 timesteps across four trajectories. By Equation 5.7, the low reward for this leaf results from a weighted average of the return estimates for these four trajectories, which in turn (by Equation 5.3) are computed by minimising a predictive loss over the preferences that have been obtained over those trajectories. We can use this inverse reasoning to answer an explanatory query: “why has this leaf been assigned a lower reward than its sibling (leaf 2 of the subtree)?” Or in more grounded terms: “in states where `dist` < 43.3, why has the tree learnt to give lower reward when `alt` < 50.2?” We seek an answer that identifies individual preferences that influence the two leaves’ reward predictions.

Recall from Section 5.2 that a trajectory preference dataset can be understood as a directed graph with edges pointing to preferred trajectories. One route to answering the preceding question is to filter this graph for preferences that directly compare a trajectory that visits leaf

1 to a trajectory that visits leaf 2. [d](#) shows that 49 such comparisons exist, and in all cases, the preference (given by the oracle) is in favour of the trajectory in leaf 2. Each of these 49 preferences contributes some amount to the difference in reward predictions, but some may be of special interest. For example, we may highlight trajectories that appear twice (e.g. ξ^{28} is preferred to both ξ^{18} and ξ^{48}), or trajectories with low overall return estimates that are nonetheless preferred to one in leaf 1 (e.g. $\xi^{43} \succ \xi^{21}$ or $\xi^{56} \succ \xi^{47}$). However, a more principled way to identify the most influential preferences is to selectively reverse each of the 49 preferences in [d](#) (e.g. $\xi^{28} \succ \xi^{18}$ becomes $\xi^{28} \prec \xi^{18}$), re-run the trajectory return estimation stage each time, and measure the change in reward (via Equation 5.7) that is given to both leaves.

The results of this preference reversal analysis are plotted in [e](#). The red and blue lines respectively indicate the leaf 1 and 2 rewards that result from each of the 49 modified preference graphs, ordered left-to-right from the preference whose reversal has the least effect on reducing the gap between rewards ($\xi^{72} \succ \xi^{48}$) to the one whose effect is greatest ($\xi^{89} \succ \xi^{18}$). No single preference reversal is enough to reduce the gap by a large amount, but this analysis provides a basis for singling out $\xi^{89} \succ \xi^{18}$ as the most influential one. Its reversal closes the gap from 2.23 to 2.01; a 9.85% reduction. Therefore, an incomplete but maximally informative single-preference answer to the explanatory query can be expressed as follows:

“In states where `dist` < 43.3, the tree gives lower reward when `alt` < 50.2 because the oracle prefers ξ^{89} to ξ^{18} (accounts for 9.85% of the gap).”

This basic method for explaining preference-based reward could be extended in various ways. For instance, it does not consider the group effect of reversing several preferences at a time, which could be handled by adopting the Shapley value framework [166, 227]. It also ignores the second-order effects of reversing preferences other than the 49 ‘direct’ ones between trajectories that visit leaves 1 and 2, although this may yield less intuitive explanations.

6.6 Explaining Model-based Action Selection

In this final section, we outline a method that exploits a powerful synergy between reward trees and model-based PETS agents. It is unique within this thesis in providing truly *intrinsic* interpretability with respect to the inner workings of a particular agent model, as opposed to a post hoc, model-agnostic, approximation of behaviour.

Recall from Section 6.3.3 that a PETS agent uses iterative planning to select its action a in state s . On each iteration, a set of H_p -step candidate action sequences is sampled from an independent Gaussian. Each action sequence is passed through a learnt dynamics model T' , which predicts a corresponding sequence of H_p future states after s . All predicted (state, action, next state) transitions are passed to the reward function, and the resultant rewards are summed (via a discount factor γ) to estimate the H_p -step return. The action sequences that produce the highest returns are identified as ‘elites’ and the mean and variance of the sampling

Gaussian are updated to more closely match the elite distribution. Over many iterations, this process should converge to a narrow sampling distribution that generates high-return action sequences, from which only the first action is sampled as the action to execute a .

This multi-step planning process, which involves the explicit consideration of counterfactual actions, provides fertile ground for interpretability methods. In combination with reward trees, the potential for explanatory decomposition is even greater. To understand this, consider how a reward tree processes all transitions predicted by T' for a particular action sequence to yield the H_p -step return (assume, as in all experiments in this chapter, that the discount factor $\gamma = 1$). There are four simple steps: apply the feature function ϕ to obtain the feature vector for each transition, count the number of feature vectors that lie within each leaf, multiply the count by that leaf’s reward, and sum over the tree. Hence, the sole factor that determines the return of an action sequence (and thus its chance of being included in the elite set) is the number of visits to each leaf that it induces. Averaging across all action sequences in each iteration of planning, we should expect later iterations to contain more visits to high-reward leaves and fewer visits to low-reward leaves, with the changing distribution of leaf visitation providing insight into the factors that influence the agent’s final choice of action.

In Figure 6.12, this is indeed what we observe. Here, a PETS agent is deployed on the Chase task using the 17-leaf reward tree from the preceding section. (a) visualises the environment state in which the agent is planning its next action, and (b) shows how the sampling distribution for that action (i.e. the first in the $H_p = 10$ -step sequence) converges over 50 planning iterations.⁵ In the state shown, EJ is separated from RJ by `dist` = 40, and although its altitude is higher than RJ, it is below the safe threshold of `alt` = 50. The action distribution converges towards pitching sharply upwards, with a slight rightwards roll. We wish to exploit the decompositional structure of the reward tree to understand *why* the agent converges to this action. As a starting point, (c) shows how the distribution of leaf visitation changes over the planning iterations. The prediction made in the previous paragraph is realised: the dominant shift is from low-reward leaves 1 and 7 to medium/high-reward leaves 8 and 9. Visits to leaves 11 and 15 temporarily increase in intermediate planning iterations, but die back down.

Although intermediate dynamics such as those of leaves 11 and 15 may be important for detailed debugging, a simple way to understand the net effect of planning is to compare the first iteration (where the action sampling distribution is very wide, generating uninformed random behaviour) to the last (where the distribution has converged to the final up-and-right turn). Figure 6.12 (d) and (e) focus on these first and last iterations, using stacked bar charts to give a ‘doubly-decomposed’ representation of the average return of each. The first level of decomposition is by planning timestep: the thick black lines show the expected reward for each of the $H_p = 10$ steps into the future. The second level of decomposition is by leaf: the heights of the coloured rectangles reflect the contribution of each leaf to the per-timestep reward.

⁵This scatter plot shows only two of the four action dimensions: roll and pitch. The other two action dimensions (yaw and thrust) also converge but are less important for understanding this particular case.

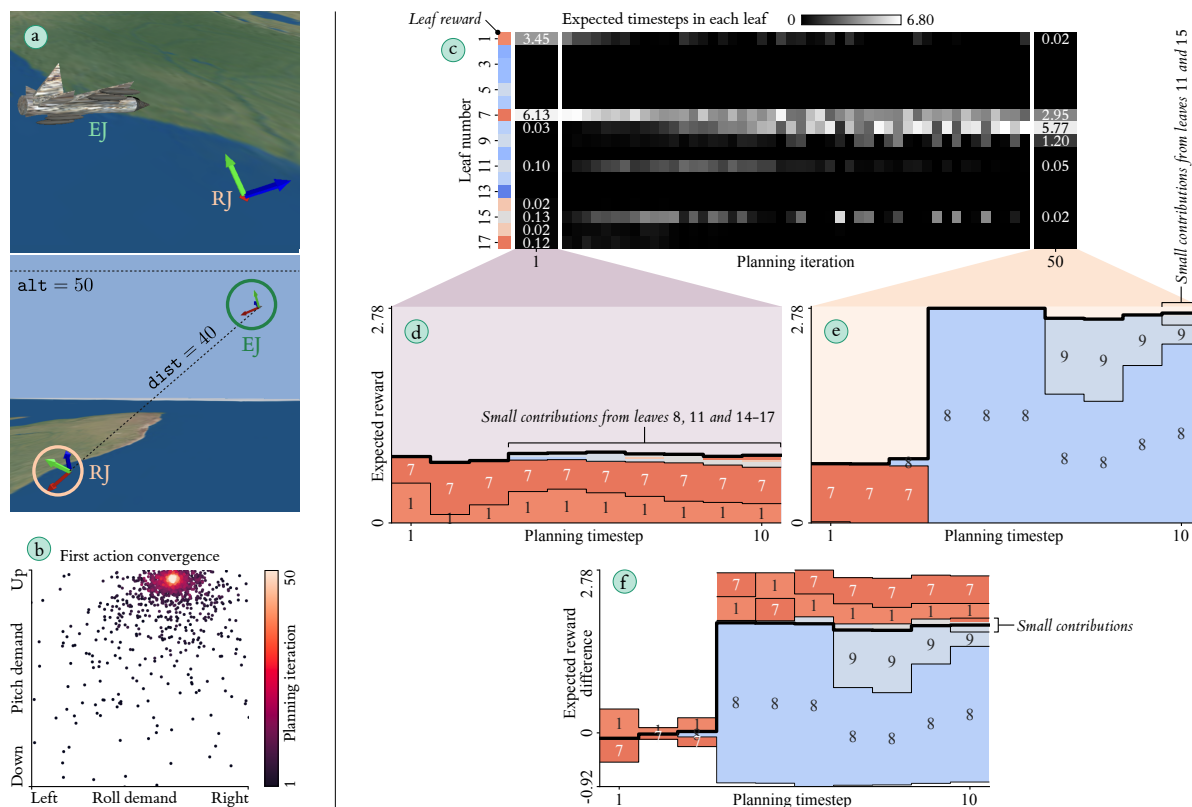


Figure 6.12: Explaining PETS agent actions with doubly-decomposed reward differences.

When the sources of expected return can be decomposed in this way, Juozapaitis et al. [134] coin the term *reward difference explanation* (RDX) to denote taking the difference between the returns of two alternative cases by component-wise subtraction, and using this to give reasons why an agent prefers one over the other.⁶ The final subplot (f) shows a visual RDX between the first and last planning iterations. The visualisation has the form of a *force plot* [167],⁷ in which the thick black line represents the overall difference in reward for each timestep. Leaves with positive contributions for that timestep ‘push the line up’ from below, and those with negative contributions ‘push it down’ from above. Since (due to the sign normalisation in Section 6.3.1) all leaves in this tree have positive reward, and leaf visitation probabilities in each timestep must sum to 1, a gain in reward from one leaf is always associated with a loss in reward from another. This means that every timestep has at least one leaf on each side of the black line.

The RDX force plot effectively answers the explanatory query “why is the average return of the final planning iteration higher than the first planning iteration?”, or in more grounded

⁶In [134], the two cases are alternative next actions to take in a particular state, and predictions are made by a decomposed state-action value (Q) network rather than explicit planning. Other work has used a similar approach to analyse tradeoffs between different planning outcomes [31, 238]. Separately, [224] use temporal decomposition to compare best- and worst-case realisations of a stochastic dynamics function. None of these prior works considers doubly-decomposed reward differences by both reward source (i.e. leaf) and timestep.

⁷The term “force plot” is not used in [167], but is in the open source code at <https://github.com/shap/shap>.

terms, “why does the agent prefer to make an up-and-right turn rather than some other random action sequence?” It can be interpreted as follows. Reading left-to-right, we find that timesteps 1 to 3 are not major drivers of the preference. Although the final iteration obtains slightly more reward from leaf 7 (below the line) and slightly less from leaf 1 (above the line), these changes balance out to have a small (in fact, slightly negative) net effect. Instead, the key advantage of the up-and-right turn is realised from timestep 4 onwards, from which the agent predicts it will obtain greatly increased reward from (initially) leaf 8 and (later on) leaf 9, instead of remaining in either leaf 1 or 7. Referring back to the tree diagram in Figure 6.8, the key factor that differentiates leaves 8 and 9 from 1 and 7 is that they are above the critical `alt = 50` threshold. In short: the agent chooses to pitch sharply upwards (and slightly rightwards) because in four timesteps’ time, this will mean it no longer violates the safe altitude threshold.

This method of exploiting the reward tree structure to analyse and explain a PETS agent’s planning process incurs very little computational overhead, since it uses data (sampled actions, predicted future states and rewards) that are generated by default as a by-product of planning. The explanations it produces, based on expected future visitation to each leaf of the tree, are contingent on outputs of the agent’s learnt dynamics model T' , and thus may reflect a somewhat inaccurate picture of the true effect of agent actions on environment states. Although this possible lack of faithfulness to the real dynamics may initially seem problematic, it is important to note that expected visitation probabilities are a faithful depiction of what the agent *believes* will happen in the future, and thus of the causal mechanism that actually drives the planning process. This is, in fact, what we want to understand when we ask *why* a given action is selected. This distinction between explanations that are ‘true to the model’ (i.e. the agent’s beliefs) and ‘true to the data’ (i.e. the real dynamics) is an important issue in interpretability research [42].

6.7 Conclusion

In this chapter, we have applied reward tree learning to the industrially-motivated use case of fast jet aircraft handling. Through extensive experiments with oracle preferences for several tasks, we showed that reward trees with around 20 leaves can achieve quantitative and qualitative performance close to that of standard NN-based approaches. We obtained evidence that the NN-tree gap reduces as the ground truth reward becomes more nonlinear and remains stable or reduces further in the presence of limited or corrupted data. To achieve these positive results, we made several methodological changes, including adopting an improved tree splitting criterion and using model-based agents for policy learning. We also explored several new interpretability directions, including one that directly exploits the planning process of a model-based agent to explain how a reward tree influences action selection. The models and experiments in this chapter have several limitations, which create opportunities for further work:

- While reward trees enabled competent policy learning on all tasks, and our new split criterion delivered a clear benefit, the performance of NN-based models was somewhat

better overall. Part of this gap may be due to inherent limits on the expressiveness of reasonably-sized trees, but the rest may be closed by further algorithmic improvements.

- The aircraft handling tasks were more complex and high-dimensional than those in the previous chapter, but simple compared with the challenges faced by real pilots. It would be interesting to extend the method to handle much longer-horizon tasks (e.g. a search and rescue mission) by combining local reward trees learnt for elementary subtasks. The *reward machines* formalism [127] may be a good way of specifying the conditions under which an agent should switch from one reward tree to another.
- A central assumption of reward learning is that manual reward specification is unreliable because human preferences are tacit. However, this may be overly pessimistic in domains with highly-trained experts like aviation. We believe that the rule-based structure of reward trees naturally lends itself to a hybrid approach of partial manual specification and partial feedback-driven learning. Implementing this would be a natural next step.
- The experiments in this chapter used synthetic oracle preferences to enable scalable quantitative evaluation. We examined the performance impacts of preference noise and myopia, but this was an imperfect proxy for experiments with real aviation experts. Such experiments would be needed to fully understand the method’s practical efficacy.

On the latter point, experiments with real experts would likely reveal that their preferences differ to some extent. While it would be easy to combine their preference data to learn an average reward tree, an alternative would be to learn an individual reward tree for each expert, and then leverage the intrinsic interpretability of these models to identify points of agreement and disagreement. This suggests an as-yet-unexplored application of reward tree learning: as a mediator for discussing and resolving divergent human preferences.

Chapter 7

Conclusions and Further Work

7.1 Review of Contributions

As AI agents play an increasing role in the modern world and their learning-based algorithms become increasingly complex, interpretability methods are essential for ensuring that their behaviour is safe, reliable and aligned with human preferences. While a growing number of techniques exist, not all are suitable for understanding the full complexity of the dynamic interaction between agents, environments and objectives. In this thesis, we have attempted to take a holistic perspective on the agent interpretability challenge through a strategy of ‘unified diversity’. Our approach was unified because we maintained a common language of abstraction with tree-based models. However, it was also diverse because we focused on various aspects of agents, including their actions, dynamics, learning and objectives.

After introducing the agent interpretability problem in Chapter 1, we formalised the tree abstraction approach in Chapter 2 and investigated its foundations. Chapter 3 proposed tree models for understanding the behaviour of an agent not just in terms of its state-to-action policy, but also via its value function and dynamics. Chapter 4 further developed the concept of trees as dynamics models, using a contrastive objective to learn models of the changes that occur during agent learning. Chapter 5 switched perspective to use trees as models of human preferences over agent behaviour, which in turn could be used by agents to learn policies that are both aligned and interpretable. Chapter 6 further refined this method and evaluated it in an industrially-motivated use case. Alongside the models themselves, we have contributed a range of transferable tools for visualisation and generating textual explanations.

7.2 Practical Uses of Proposed Models

In Section 1.1, we introduced six user stories to illustrate why various stakeholders require agents to be interpretable. In the following, we briefly consider how our models might assist these stakeholders, highlighting how they could be combined in complementary ways.

- **As an owner of an AI agent, I want to** verify its capabilities and limitations in as-yet-unseen scenarios, **so that** I can deploy it with confidence.
 - Suppose that a startup has built a robot vacuum cleaner. They have data on its behaviour in many test rooms and wish to know if it is ready for use in customers' homes. They could use the test data to grow a TRIPLETREE model. Through projected hyperrectangle plots of the robot's policy and value function, they could establish trends in the robot's actions and performance across environment states. They could also use hypothetical trajectories to assess the robot's ability to move between certain start and goal conditions (e.g. from under a kitchen table to its charging station) and the most likely routes to get there. In this way, they could establish features of customer homes that pose challenges to the current design.
- **As a user of an AI agent, I want to** understand how my actions are influencing its behaviour, **so that** I gain some control over the behaviour produced in future.
 - Future medical professionals may collaborate with automated agents during treatment planning. As non-experts in AI, doctors and nurses might best understand such an agent's recommendations through textual explanations. By learning a tree-structured model to approximate the agent's policy, it would be possible to explain actions factually and counterfactually in natural language and construct temporal explanations of entire treatment plans. By expressing preferences over alternative plans, the medics could contribute to improving the agent itself via reward tree learning, with reward predictions always being traceable back to individual preference labels. If the agent used explicit model-based planning, it could even explain its actions directly in terms of the influence of different reward tree leaves.
- **As a regulator of technology, I want to** screen proposed AI agents for pernicious functionality and biases, **so that** I can protect the public from harm.
 - Consider a financial regulator tasked with approving or blocking an RL-based algorithmic trading agent. Rather than requiring the firm that developed the agent to release all their sensitive data, the regulator may request that the agent's learning history be summarised in a sequence of abstract transition models via our CSTA algorithm. This would allow them to review the behaviours explored during learning. They could then determine if the agent has gained sufficient experience of, and robustness to, adverse market events, and has unlearned any unsafe behaviours. The agent may have been guided by feedback from expert human traders. If this was done using reward tree learning, the firm could also release the sequence of reward trees used throughout the learning process. This would help the regulator to ascertain not only which behaviours occurred during learning, but why they occurred.

- **As an AI practitioner, I want to** trace the effects of changes I make to an agent model and its learning algorithm, **so that** I can make improvements efficiently.
 - This practitioner may be training an agent for a challenging exploration problem, such as mapping deep sea caves in an AUV. Once again, the CSTA method could be valuable. It could reveal if learning leads the agent to become stuck in certain types of state, and even how this varies between training runs with different agent hyperparameters. For moments in training when rapid changes occur, the practitioner may wish to load up a checkpointed policy and learn a tree model to approximate it. Consulting the diagram of this tree could provide further insight into what, if anything, might be going wrong, including whether the agent responds in unintuitive ways to features of the environment. This may inspire changes to the agent’s internal structure or how it represents the environment state.
- **As an AI researcher, I want to** develop scientific insight into the mechanisms and emergent properties of agent learning, **so that** this can inform future breakthroughs.
 - Suppose this researcher is using a video game benchmark to understand the capabilities of a state-of-the-art RL agent. All of our models could be brought to bear on this effort, and experts would be well-equipped to integrate them in informative ways. Multiattribute visualisations of a TRIPLETREE model, and transition graphs produced by CSTA, could help to reveal both global trends and local sensitivities in the agent’s policy, value function and state dynamics. The researcher could compare these results to models learnt from the gameplay data of humans of varying levels of expertise. This could establish where the RL agent exhibits human-like strategies and deficiencies, and where its behaviour is genuinely novel.
- **As a person living in a world with ever more complex and numerous AI agents, I want to** know whether their behaviour and learning are aligned with my best interests and those of my community, **so that** I maintain an ability to control them.
 - Consider a future driverless family car. Each family member can be assigned a unique reward tree, learnt from their feedback on alternative driving styles, which influences the car’s objective (and thus its behaviour) when they are a passenger. Perhaps one child strongly dislikes speed bumps and prioritises going gently across them, while one parent prefers to take scenic routes over rushing to their destination. Each person can trust that journeys will align with their preferences because the car can verbally justify its driving decisions in terms of their personalised rule structure. When the whole family is in the car, a compromise driving style may be sought by comparing their individual trees, retaining points of agreement, and asking for clarification on points of disagreement.

7.3 Successes of the Overall Approach

In Sections 1.4 (T1) and 2.6, we discussed how impactful interpretability work often proposes simple, generic algorithm schemas based on clear assumptions and principles. By motivating the flexible strategy of tree abstraction and following it throughout, we have attempted to do the same. Framing tree abstraction in generic terms allowed us to think beyond how it has been used before. In particular, we are aware of no prior examples of trees that jointly represent agent actions, value functions and state dynamics, no prior examples of trees optimised to reveal change points in agent learning, and no prior examples of trees used in reward learning. All of this work was novel while being grounded in a model class familiar to the interpretability community, and using well-understood algorithms for tree growth and pruning. The common language of trees has enabled the transfer of analysis techniques across chapters (e.g. counterfactuals, hyperrectangle projection plots, leaf visitation heatmaps), implying that the various methods could be readily combined into a unified software package. They could also be integrated more directly. In the spirit of TRIPLETREE, it would be fairly simple to develop a hybrid model providing *all* of the functionality of the individual models by combining their split criteria. For some application contexts, this may be preferable to multiple specialist models.

7.4 Limitations of the Overall Approach

As discussed in Section 1.3.1, interpretability is hard to define, but many methods follow the heuristic of enforcing or approximating simplicity. In this thesis, we arrived at the tree abstraction approach by defining simplicity via the principle of abstraction, the desiderata of convexity, partitioning, hierarchy and axis-alignment, and the property of query-dependent efficiency. In doing so, we left many equally valid routes unexplored and accepted an unavoidable path dependence that constrained our thinking around all concrete methods and analyses. If we had instead opted to use linear models or naïve Bayes classifiers as our common representation, we would have doubtless had very different insights and results.

On several occasions, we have alluded to differentiable trees that can be learnt by gradient-based methods by sacrificing axis-alignment. These models are popular in the agent interpretability literature (e.g. [51, 135, 192, 230, 242]), They can be better integrated with machine learning hardware and software than the more classical approach of searching over discrete splits. They can also model a larger class of functions, so they would probably outperform our current methods on quantitative metrics. We have not explored differentiable trees because we believe (based on a continual review of existing work) that they induce a significant reduction in interpretability. However, it is plausible that they provide a more scalable long-term solution for interpreting agents, especially if they could be reliably simplified through techniques such as sparsity regularisation. Our methods could be adapted to use differentiable trees with few conceptual changes, but there is also space in the literature for both model classes to flourish.

7.5 Further Work

In addition to the concrete further work directions identified in each chapter-specific conclusion (Sections 3.11, 4.8, 5.10 and 6.7), we can envisage several larger research programmes to build on the ideas explored in this thesis. Each of these could form a PhD topic in itself.

User Evaluation We have combined quantitative metrics and careful qualitative interpretation to evaluate our proposed methods. However, as noted in Section 1.3.1, the value of an interpretability method can only be fully assessed through experiments with real human users. Work in this direction would require the development of user interfaces for customising and visualising the models. We would be especially interested in assessing whether the interpretability of reward tree learning helps users to provide better feedback and ultimately improve alignment.

Interpretable Feature Generation From the caveat in Section 2.3.4 onwards, we have acknowledged the need for tree models to use basic features that are individually interpretable (i.e. grounded in natural language concepts). The fact that these are not always readily available could be the main limiting factor to our methods' adoption. While the operator-based approach in Section 3.2.2 provides a starting point, more work must be done to develop scalable feature generation methods for arbitrary environments, including those with image-based observations.

Optimal and Online Algorithms Throughout this thesis, we have used greedy tree growth algorithms, which optimise over one split at a time rather than an entire tree. The quantitative performance of the models (and thus their efficiency as summaries of agents and environments) may be enhanced by using globally optimal tree induction algorithms [25]. Also, aside from in Chapter 5, all algorithms grow trees from scratch on fixed datasets. Many future applications of interpretability could involve monitoring agents in real-time, so extending all methods to better handle online streaming data would be beneficial.

Additional Models There is scope for much more work to develop variants on the theme of tree abstraction for agent interpretability, including those that take an intrinsic or mechanistic perspective (see Section 1.3.2 for definitions). Examples may include:

- Model-based agents with trees as their dynamics models, expressible as transition graphs.
- Mechanistic tree models of the internal activations of an agent's policy network.
- Tree abstractions of an agent's action space to understand when and why certain types of action are taken.

Multiagent Applications As AI becomes more prevalent in society, increasingly many systems will involve dynamic interactions between multiple agents (and multiple humans). We see multiagent systems as the next great frontier of interpretability research. It would be valuable to explore how tree abstraction could contribute to this effort.

Bibliography

- [1] S. AGARWAL, C. HERRMANN, G. WALLNER, AND F. BECK, *Visualizing ai playtesting data of 2d side-scrolling games*, in 2020 IEEE Conference on Games (CoG), 2020, pp. 572–575.
- [2] S. AGARWAL, G. WALLNER, AND F. BECK, *Bombalytics: Visualization of competition and collaboration strategies of players in a bomb laying game*, Computer Graphics Forum, 39 (2020), pp. 89–100.
- [3] E. ALBINI, S. SHARMA, S. MISHRA, D. DERVOVIC, AND D. MAGAZZENI, *On the connection between game-theoretic feature attributions and counterfactual explanations*, in Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, 2023.
- [4] A. ALSHEHRI, T. MILLER, AND M. VERED, *Explainable goal recognition: A framework based on weight of evidence*, arXiv preprint arXiv:2303.05622, (2023).
- [5] S. AMINIKHANGHAHI AND D. J. COOK, *A survey of methods for time series change point detection*, Knowledge and information systems, 51 (2017), pp. 339–367.
- [6] 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.
- [7] O. AMIR, F. DOSHI-VELEZ, AND D. SARNE, *Summarizing agent strategies*, Autonomous Agents and Multi-Agent Systems, 33 (2019), p. 628–644.
- [8] Y. AMITAI AND O. AMIR, *“I Don’t Think So”: Summarizing Policy Disagreements for Agent Comparison*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 5269–5276.
- [9] Y. AMITAI, G. AVNI, AND O. AMIR, *Asq-it: Interactive explanations for reinforcement-learning agents*, arXiv preprint arXiv:2301.09941, (2023).
- [10] R. M. ANNASAMY AND K. SYCARA, *Towards better interpretability in deep q-networks*, in Proceedings of the AAAI conference on artificial intelligence, vol. 33, 2019, pp. 4561–4569.

- [11] S. ARADI, *Survey of deep reinforcement learning for motion planning of autonomous vehicles*, IEEE Transactions on Intelligent Transportation Systems, 23 (2020), pp. 740–759.
- [12] S. ARMSTRONG, J. LEIKE, L. ORSEAU, AND S. LEGG, *Pitfalls of learning a reward function online*, in Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, 2020.
- [13] C. ARZATE CRUZ AND T. IGARASHI, *A survey on interactive reinforcement learning: Design principles and open challenges*, in Proceedings of the 2020 ACM Designing Interactive Systems Conference, 2020, pp. 1195–1209.
- [14] A. ATREY, K. CLARY, AND D. JENSEN, *Exploratory not explanatory: Counterfactual analysis of saliency maps for deep reinforcement learning*, in International Conference on Learning Representations, 2020.
- [15] A. T. AZAR, A. KOUBAA, N. ALI MOHAMED, H. A. IBRAHIM, Z. F. IBRAHIM, M. KAZIM, A. AMMAR, B. BENJDIRA, A. M. KHAMIS, I. A. HAMEED, ET AL., *Drone deep reinforcement learning: A review*, Electronics, 10 (2021), p. 999.
- [16] B. BACH, P. DRAGICEVIC, D. ARCHAMBAULT, C. HURTER, AND S. CARPENDALE, *A review of temporal data visualizations based on space-time cube operations*, in Eurographics conference on visualization, 2014.
- [17] A. BAJCSY, D. P. LOSEY, M. K. O’MALLEY, AND A. D. DRAGAN, *Learning robot objectives from physical human interaction*, in Conference on Robot Learning, PMLR, 2017, pp. 217–226.
- [18] A. BAKER, *Simplicity*, in The Stanford Encyclopedia of Philosophy, E. N. Zalta, ed., Metaphysics Research Lab, Stanford University, Summer 2022 ed., 2022.
- [19] G. BARYANNIS, S. DANI, AND G. ANTONIOU, *Predicting supply chain risks using machine learning: The trade-off between performance and interpretability*, Future Generation Computer Systems, 101 (2019), pp. 993–1004.
- [20] O. BASTANI, Y. PU, AND A. SOLAR-LEZAMA, *Verifiable reinforcement learning via policy extraction*, Advances in Neural Information Processing Systems, 31 (2018).
- [21] L. BECHBERGER AND K.-U. KÜHNBERGER, *A thorough formalization of conceptual spaces*, in Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz), Springer, 2017, pp. 58–71.
- [22] D. BEECHEY, T. M. SMITH, AND Ö. ŞİMŞEK, *Explaining reinforcement learning with shapley values*, in International Conference on Machine Learning, PMLR, 2023, pp. 2003–2014.

- [23] M. BELKAID, *Explanation through behavior: a human-inspired framework for explainable robotics*, in ICRA2023 Workshop on Explainable Robotics, 2023.
- [24] M. K. BERGMAN, *Hierarchy in knowledge systems*, Knowledge Organization, 49 (2022), pp. 40–66. Also available in ISKO Encyclopedia of Knowledge Organization, eds. Birger Hjørland and Claudio Gnoli, <https://www.isko.org/cyclo/hierarchy>.
- [25] D. BERTSIMAS AND J. DUNN, *Optimal classification trees*, Machine Learning, 106 (2017), pp. 1039–1082.
- [26] T. BEWLEY, *Am I Building a White Box Agent or Interpreting a Black Box Agent?*, arXiv preprint arXiv:2007.01187, (2020).
- [27] E. BIYIK, D. P. LOSEY, M. PALAN, N. C. LANDOLFI, G. SHEVCHUK, AND D. SADIGH, *Learning reward functions from diverse sources of human feedback: Optimally integrating demonstrations and preferences*, The International Journal of Robotics Research, 41 (2022), pp. 45–67.
- [28] A. BOBU, M. WIGGERT, C. TOMLIN, AND A. D. DRAGAN, *Inducing structure in reward learning by learning features*, The International Journal of Robotics Research, 41 (2022), pp. 497–518.
- [29] S. BOOTH, J. SHAH, S. NIEKUM, P. STONE, AND A. ALLIEVI, *The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications*, 2023.
- [30] R. A. BRADLEY AND M. E. TERRY, *Rank analysis of incomplete block designs: I. the method of paired comparisons*, Biometrika, 39 (1952), pp. 324–345.
- [31] M. BRANDAO, G. CANAL, S. KRIVIĆ, AND D. MAGAZZENI, *Towards providing explanations for robot motion planning*, in 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 3927–3933.
- [32] L. BREIMAN, J. FRIEDMAN, R. OLSHEN, AND C. STONE, *Classification and regression trees*. Wadsworth & Brooks, Cole Statistics/Probability Series, (1984).
- [33] A. BRENNEN, *What do people really want when they say they want” explainable ai?” we asked 60 stakeholders.*, in Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, 2020, pp. 1–7.
- [34] C. BREWITT, B. GYEVNAR, S. GARCIN, AND S. V. ALBRECHT, *Grit: Fast, interpretable, and verifiable goal recognition with learned decision trees for autonomous driving*, in 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2021), 2021.

- [35] S. L. BRUNTON, J. NATHAN KUTZ, K. MANOHAR, A. Y. ARAVKIN, K. MORGANSEN, J. KLEMISCH, N. GOEBEL, J. BUTTRICK, J. POSKIN, A. W. BLOM-SCHIEBER, ET AL., *Data-driven aerospace engineering: reframing the industry with machine learning*, AIAA Journal, 59 (2021), pp. 2820–2847.
- [36] S. CAO, X. WANG, R. ZHANG, H. YU, AND L. SHEN, *From demonstration to flight: Realization of autonomous aerobatic maneuvers for fast, miniature fixed-wing uavs*, IEEE Robotics and Automation Letters, 7 (2022), pp. 5771–5778.
- [37] Z. CAO, K. WONG, AND C.-T. LIN, *Weak human preference supervision for deep reinforcement learning*, IEEE Transactions on Neural Networks and Learning Systems, 32 (2021), pp. 5369–5378.
- [38] A. CARPENTIER, A. LAZARIC, M. GHAVAMZADEH, R. MUNOS, AND P. AUER, *Upper-confidence-bound algorithms for active learning in multi-armed bandits*, in International Conference on Algorithmic Learning Theory, Springer, 2011, pp. 189–203.
- [39] D. V. CARVALHO, E. M. PEREIRA, AND J. S. CARDOSO, *Machine learning interpretability: A survey on methods and metrics*, Electronics, 8 (2019), p. 832.
- [40] S. CASPER, T. RAUKER, A. HO, AND D. HADFIELD-MENELL, *Sok: Toward transparent AI: A survey on interpreting the inner structures of deep neural networks*, in First IEEE Conference on Secure and Trustworthy Machine Learning, 2023.
- [41] V. CHARISI, N. DÍAZ-RODRÍGUEZ, B. MAWHIN, AND L. MERINO, *On children’s exploration, aha! moments and explanations in model building for self-regulated problem-solving*, in IJCAI-ECAI workshop on AI evaluation beyond metrics, 2022.
- [42] H. CHEN, J. D. JANIZEK, S. LUNDBERG, AND S.-I. LEE, *True to the model or true to the data?*, arXiv preprint arXiv:2006.16234, (2020).
- [43] N. CHOMSKY, *A review of BF Skinner’s Verbal Behavior*, Language, 35 (1959), pp. 26–58.
- [44] P. F. CHRISTIANO, J. LEIKE, T. BROWN, M. MARTIC, S. LEGG, AND D. AMODEI, *Deep reinforcement learning from human preferences*, Advances in Neural Information Processing Systems, 30 (2017).
- [45] K. CHUA, R. CALANDRA, R. MCALLISTER, AND S. LEVINE, *Deep reinforcement learning in a handful of trials using probabilistic dynamics models*, Advances in Neural Information Processing Systems, 31 (2018).
- [46] G. CIATTO, M. I. SCHUMACHER, A. OMICINI, AND D. CALVARESI, *Agent-based explanations in ai: Towards an abstract framework*, in International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems, Springer, 2020, pp. 3–20.

- [47] P. CICHOSZ AND L. PAWELCZAK, *Imitation learning of car driving skills with decision trees and random forests*, International Journal of Applied Mathematics and Computer Science, 24 (2014), pp. 579–597.
- [48] S. G. CLARKE AND I. HWANG, *Deep reinforcement learning control for aerobatic maneuvering of agile fixed-wing aircraft*, in AIAA Scitech 2020 Forum, 2020, p. 0136.
- [49] L. C. COBO, C. L. ISBELL JR, AND A. L. THOMAZ, *Automatic task decomposition and state abstraction from demonstration*, Georgia Institute of Technology, 2012.
- [50] M. COHN, *User stories applied: For agile software development*, Addison-Wesley Professional, 2004.
- [51] Y. COPPENS, K. EFTHYMIADIS, T. LENAERTS, AND A. NOWÉ, *Distilling deep reinforcement learning policies in soft decision trees*, in IJCAI/ECAI Workshop on Explainable Artificial Intelligence, 2019.
- [52] Y. COPPENS, D. STECKELMACHER, C. M. JONKER, AND A. NOWÉ, *Synthesising reinforcement learning policies through set-valued inductive rule learning*, in Trustworthy AI - Integrating Learning, Optimization and Reasoning, Cham, 2021, Springer International Publishing, pp. 163–179.
- [53] E. COUMANS AND Y. BAI, *Pybullet, a python module for physics simulation for games, robotics and machine learning*.
<http://pybullet.org>, 2016–2021.
- [54] F. CRUZ, C. YOUNG, R. DAZELEY, AND P. VAMPLEW, *Evaluating Human-like Explanations for Robot Actions in Reinforcement Learning Scenarios*, in International Conference on Intelligent Robots and Systems (IROS), 2022.
- [55] L. CSATÓ, *A graph interpretation of the least squares ranking method*, Social Choice and Welfare, 44 (2015), pp. 51–69.
- [56] J. CULLEN AND A. BRYMAN, *The knowledge acquisition bottleneck: time for reassessment?*, Expert Systems, 5 (1988), pp. 216–225.
- [57] N. DAHLIN, K. C. KALAGARLA, N. NAIK, R. JAIN, AND P. NUZZO, *Designing interpretable approximations to deep reinforcement learning*, arXiv preprint arXiv:2010.14785, (2020).
- [58] Y. DAI, Q. CHEN, J. ZHANG, X. WANG, Y. CHEN, T. GAO, P. XU, S. CHEN, S. LIAO, H. JIANG, ET AL., *Enhanced oblique decision tree enabled policy extraction for deep reinforcement learning in power system emergency control*, Electric Power Systems Research, 209 (2022), p. 107932.

- [59] 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.
- [60] O. DAVOODI AND M. KOMEILI, *Feature-based interpretable reinforcement learning based on state-transition models*, in 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2021, pp. 301–308.
- [61] R. DAZELEY, P. VAMPLEW, AND F. CRUZ, *Explainable reinforcement learning for broad-xai: a conceptual framework and survey*, Neural Computing and Applications, (2023), pp. 1–24.
- [62] J. DE LEEUW AND P. MAIR, *Multidimensional scaling using majorization: Smacof in r*, Journal of statistical software, 31 (2009), pp. 1–30.
- [63] G. DE'ATH, *Multivariate regression trees: a new technique for modeling species–environment relationships*, Ecology, 83 (2002), pp. 1105–1117.
- [64] J. DEGRAVE, F. FELICI, J. BUCHLI, M. NEUNERT, B. TRACEY, F. CARPANESE, T. EWALDS, R. HAFNER, A. ABDOLMALEKI, D. DE LAS CASAS, ET AL., *Magnetic control of tokamak plasmas through deep reinforcement learning*, Nature, 602 (2022), pp. 414–419.
- [65] Q. DELFOSSE, H. SHINDO, D. DHAMI, AND K. KERSTING, *Interpretable and explainable logical policies via neurally guided symbolic abstraction*, arXiv preprint arXiv:2306.01439, (2023).
- [66] S. V. DESHMUKH, A. DASGUPTA, B. KRISHNAMURTHY, N. JIANG, C. AGARWAL, G. THEOCHAROUS, AND J. SUBRAMANIAN, *Explaining RL decisions with trajectories*, in The Eleventh International Conference on Learning Representations, 2023.
- [67] S. V. DESHMUKH, S. R., S. VIJAY, J. SUBRAMANIAN, AND C. AGARWAL, *Counterfactual explanation policies in rl*, in ICML Workshop on Counterfactuals in Minds and Machines, 2023.
- [68] S. DESHPANDE, B. EYSENBACH, AND J. SCHNEIDER, *Interactive visualization for debugging rl*, arXiv preprint arXiv:2008.07331, (2020).
- [69] R. DEVIDZE, G. RADANOVIC, P. KAMALARUBAN, AND A. SINGLA, *Explicable reward design for reinforcement learning agents*, Advances in Neural Information Processing Systems, 34 (2021), pp. 20118–20131.
- [70] D. DEWEY, *Reinforcement learning and the reward engineering principle*, in 2014 AAAI Spring Symposium Series, 2014.

-
- [71] Y. DHEBAR, K. DEB, S. NAGESHRAO, L. ZHU, AND D. FILEV, *Interpretable-ai policies using evolutionary nonlinear decision trees for discrete action systems*, arXiv preprint arXiv:2009.09521, (2020).
- [72] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, *Numerische mathematik*, 1 (1959), pp. 269–271.
- [73] Z. DING, P. HERNANDEZ-LEAL, G. W. DING, C. LI, AND R. HUANG, *Cdt: Cascading decision trees for explainable reinforcement learning*, arXiv preprint arXiv:2011.07553, (2020).
- [74] F. DOSHI-VELEZ AND B. KIM, *Towards a rigorous science of interpretable machine learning*, arXiv preprint arXiv:1702.08608, (2017).
- [75] N. DOUGLAS, D. YIM, B. KARTAL, P. HERNANDEZ-LEAL, F. MAURER, AND M. E. TAYLOR, *Towers of saliency: A reinforcement learning visualization using immersive environments*, in *Proceedings of the 2019 ACM International Conference on Interactive Surfaces and Spaces*, 2019, pp. 339–342.
- [76] P. DU, S. MURTHY, AND K. DRIGGS-CAMPBELL, *Conveying autonomous robot capabilities through contrasting behaviour summaries*, arXiv preprint arXiv:2304.00367, (2023).
- [77] J. W. DUNN, *Optimal trees for prediction and prescription*, PhD thesis, Massachusetts Institute of Technology, 2018.
- [78] R. DUTTA, Q. WANG, A. SINGH, D. KUMARJIGUDA, L. XIAOLI, AND S. JAYAVELU, *S-reinforce: A neuro-symbolic policy gradient approach for interpretable reinforcement learning*, arXiv preprint arXiv:2305.07367, (2023).
- [79] S. DŽEROSKI, L. D. RAEDT, AND H. BLOCKEEL, *Relational reinforcement learning*, in *International Conference on Inductive Logic Programming*, Springer, 1998, pp. 11–22.
- [80] J. EARLY, T. BEWLEY, C. EVERS, AND S. RAMCHURN, *Non-markovian reward modelling from trajectory labels via interpretable multiple instance learning*, *Advances in Neural Information Processing Systems*, 35 (2022), pp. 27652–27663.
- [81] A. ERASMUS, T. D. BRUNET, AND E. FISHER, *What is interpretability?*, *Philosophy & Technology*, (2020), pp. 1–30.
- [82] B. ERIC, N. FREITAS, AND A. GHOSH, *Active Preference Learning with Discrete Choice Data*, *Advances in Neural Information Processing Systems*, 20 (2007).

BIBLIOGRAPHY

- [83] B. EYSENBACH, A. GUPTA, J. IBARZ, AND S. LEVINE, *Diversity is all you need: Learning skills without a reward function*, in International Conference on Learning Representations, 2019.
- [84] J. F, *Transparent value alignment*, in Companion of the 2023 ACM/IEEE International Conference on Human-Robot Interaction, 2023, pp. 557–560.
- [85] J. B. FADDOUL, B. CHIDLOVSKII, R. GILLERON, AND F. TORRE, *Learning multiple tasks with boosted decision trees*, in Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2012, Bristol, UK, September 24–28, 2012. Proceedings, Part I 23, Springer, 2012, pp. 681–696.
- [86] FARAMA FOUNDATION, *Gymnasium*.
<https://github.com/Farama-Foundation/Gymnasium>, 2023.
- [87] M. FINKELSTEIN, N. S. LEVY, L. LIU, Y. KOLUMBUS, D. C. PARKES, J. ROSENSCHEIN, AND S. KEREN, *Explainable reinforcement learning via model transforms*, in Advances in Neural Information Processing Systems, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, eds., 2022.
- [88] B. L. FREDRICKSON AND D. KAHNEMAN, *Duration neglect in retrospective evaluations of affective episodes.*, Journal of personality and social psychology, 65 (1993), p. 45.
- [89] S. FUJIMOTO, H. HOOF, AND D. MEGER, *Addressing function approximation error in actor-critic methods*, in International Conference on Machine Learning, PMLR, 2018, pp. 1587–1596.
- [90] J. GAJCIN AND I. DUSPARIC, *Counterfactual explanations for reinforcement learning*, arXiv preprint arXiv:2210.11846, (2022).
- [91] ———, *Raccor: Towards reachable and certain counterfactual explanations for reinforcement learning*, arXiv preprint arXiv:2303.04475, (2023).
- [92] J. GAJCIN, R. NAIR, T. PEDAPATI, R. MARINESCU, E. DALY, AND I. DUSPARIC, *Contrastive explanations for comparing preferences of reinforcement learning agents*, in AAAI Workshop on Interactive Machine Learning, 2022.
- [93] O. GALLITZ, O. DE CANDIDO, M. BOTSCH, AND W. UTSCHICK, *Interpretable feature generation using deep neural networks and its application to lane change detection*, in 2019 IEEE Intelligent Transportation Systems Conference (ITSC), IEEE, 2019, pp. 3405–3411.
- [94] P. GÄRDENFORS, *Conceptual spaces: The geometry of thought*, MIT press, 2004.

- [95] P. GÄRDENFORS AND M. WARGLIEN, *Using conceptual spaces to model actions and events*, Journal of semantics, 29 (2012), pp. 487–519.
- [96] V. B. GJÆRUM, I. STRÜMKE, J. LØVER, T. MILLER, AND A. M. LEKKAS, *Model tree methods for explaining deep reinforcement learning agents in real-time robotic applications*, Neurocomputing, 515 (2023), pp. 133–144.
- [97] V. B. GJÆRUM, I. STRÜMKE, O. A. ALSOS, AND A. M. LEKKAS, *Explaining a deep reinforcement learning docking agent using linear model trees with user adapted visualization*, Journal of Marine Science and Engineering, 9 (2021).
- [98] C. GLANOIS, P. WENG, M. ZIMMER, D. LI, T. YANG, J. HAO, AND W. LIU, *A survey on interpretable reinforcement learning*, arXiv preprint arXiv:2112.13112, (2021).
- [99] A. GLEAVE, M. D. DENNIS, S. LEGG, S. RUSSELL, AND J. LEIKE, *Quantifying differences in reward functions*, in International Conference on Learning Representations, 2021.
- [100] 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.
- [101] S. GRIFFITH, K. SUBRAMANIAN, J. SCHOLZ, C. L. ISBELL, AND A. L. THOMAZ, *Policy shaping: Integrating human feedback with reinforcement learning*, Advances in neural information processing systems, 26 (2013).
- [102] R. GUIDOTTI, A. MONREALE, F. GIANNOTTI, D. PEDRESCHI, S. RUGGIERI, AND F. TURINI, *Factual and counterfactual explanations for black box decision making*, IEEE Intelligent Systems, (2019).
- [103] C. GULCEHRE, Z. WANG, A. NOVIKOV, T. PAINE, S. GÓMEZ, K. ZOLNA, R. AGARWAL, J. S. MEREL, D. J. MANKOWITZ, C. PADURARU, ET AL., *Rl unplugged: A suite of benchmarks for offline reinforcement learning*, Advances in Neural Information Processing Systems, 33 (2020), pp. 7248–7259.
- [104] H. GULLIKSEN, *A least squares solution for paired comparisons with incomplete data*, Psychometrika, 21 (1956), pp. 125–134.
- [105] W. GUO, X. WU, U. KHAN, AND X. XING, *EDGE: Explaining Deep Reinforcement Learning Policies*, Advances in Neural Information Processing Systems, 34 (2021).
- [106] Y. GUO, P. TIAN, J. KALPATHY-CRAMER, S. OSTMO, J. P. CAMPBELL, M. F. CHIANG, D. ERDOGMUS, J. G. DY, AND S. IOANNIDIS, *Experimental design under the bradley-terry model.*, in IJCAI, 2018, pp. 2198–2204.

- [107] 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.
- [108] B. HAMMER AND E. HÜLLERMEIER, *Interpretable machine learning: On the problem of explaining model change*, in Proceedings of Workshop Computational Intelligence, vol. 25, 2021, p. 1.
- [109] S. HARNAD, *The symbol grounding problem*, Physica D: Nonlinear Phenomena, 42 (1990), pp. 335–346.
- [110] B. HAYES AND J. A. SHAH, *Improving robot controller transparency through autonomous policy explanation*, in Proceedings of the 2017 ACM/IEEE international conference on human-robot interaction, 2017, pp. 303–312.
- [111] L. HE, N. AOUF, AND B. SONG, *Explainable deep reinforcement learning for uav autonomous path planning*, Aerospace science and technology, 118 (2021), p. 107052.
- [112] W. HE, T.-Y. LEE, J. VAN BAAR, K. WITTENBURG, AND H.-W. SHEN, *Dynamic-explorer: Visual analytics for robot control tasks involving dynamics and lstm-based control policies*, in 2020 IEEE Pacific Visualization Symposium (PacificVis), IEEE, 2020, pp. 36–45.
- [113] D. HEIN, S. UDLUFT, AND T. A. RUNKLER, *Interpretable policies for reinforcement learning by genetic programming*, Engineering Applications of Artificial Intelligence, 76 (2018), pp. 158–169.
- [114] R. HERBRICH, T. MINKA, AND T. GRAEPEL, *Trueskill: A bayesian skill rating system*, in Proceedings of the 19th international conference on neural information processing systems, 2006, pp. 569–576.
- [115] G. HESSLOW, *The problem of causal selection*, Contemporary science and natural explanation: Commonsense conceptions of causality, (1988), pp. 11–32.
- [116] J. HILTON, N. CAMMARATA, S. CARTER, G. GOH, AND C. OLAH, *Understanding rl vision*, Distill, 5 (2020), p. e29.
- [117] C. F. HOCKETT AND C. D. HOCKETT, *The origin of speech*, Scientific American, 203 (1960), pp. 88–97.
- [118] D. R. HOFSTADTER AND E. SANDER, *Surfaces and essences: Analogy as the fuel and fire of thinking*, Basic books, 2013.

-
- [119] A. HOLZINGER, G. LANGS, H. DENK, K. ZATLOUKAL, AND H. MÜLLER, *Causability and explainability of artificial intelligence in medicine*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 9 (2019), p. e1312.
- [120] X. HU, J. LI, X. ZHAN, Q.-S. JIA, AND Y.-Q. ZHANG, *Query-policy misalignment in preference-based reinforcement learning*, arXiv preprint arXiv:2305.17400, (2023).
- [121] J. HUANG, P. P. ANGELOV, AND C. YIN, *Interpretable policies for reinforcement learning by empirical fuzzy sets*, Engineering Applications of Artificial Intelligence, 91 (2020), p. 103559.
- [122] 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.
- [123] S. H. HUANG, D. HELD, P. ABBEEL, AND A. D. DRAGAN, *Enabling robots to communicate their objectives*, Autonomous Robots, 43 (2019), pp. 309–326.
- [124] T. HUBER, M. DEMMLER, S. MERTES, M. L. OLSON, AND E. ANDRÉ, *Ganterfactual-rl: Understanding reinforcement learning agents’ strategies through visual counterfactual explanations*, in Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS ’23, Richland, SC, 2023, International Foundation for Autonomous Agents and Multiagent Systems, p. 1097–1106.
- [125] 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.
- [126] B. IBARZ, J. LEIKE, T. POHLEN, G. IRVING, S. LEGG, AND D. AMODEI, *Reward learning from human preferences and demonstrations in atari*, in Advances in Neural Information Processing Systems, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds., vol. 31, 2018.
- [127] R. T. ICARTE, T. KLASSEN, R. VALENZANO, AND S. MCILRAITH, *Using reward machines for high-level task specification and decomposition in reinforcement learning*, in International Conference on Machine Learning, PMLR, 2018, pp. 2107–2116.
- [128] J. E. JACKSON AND M. FLECKENSTEIN, *An evaluation of some statistical techniques used in the analysis of paired comparison data*, Biometrics, 13 (1957), pp. 51–64.
- [129] T. JAUNET, R. VUILLEMOT, AND C. WOLF, *DrIviz: Understanding decisions and memory in deep reinforcement learning*, in Computer Graphics Forum, vol. 39, Wiley Online Library, 2020, pp. 49–61.

- [130] E. JENNER AND A. GLEAVE, *Preprocessing reward functions for interpretability*, arXiv preprint arXiv:2203.13553, (2022).
- [131] H. J. JEON, S. MILLI, AND A. DRAGAN, *Reward-rational (implicit) choice: A unifying formalism for reward learning*, Advances in Neural Information Processing Systems, 33 (2020), pp. 4415–4426.
- [132] A. JHUNJHUNWALA, J. LEE, S. SEDWARDS, V. ABDELZAD, AND K. CZARNECKI, *Improved policy extraction via online q-value distillation*, in 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020, pp. 1–8.
- [133] W.-C. JIANG, K.-S. HWANG, AND J.-L. LIN, *An experience replay method based on tree structure for reinforcement learning*, IEEE Transactions on Emerging Topics in Computing, 9 (2019), pp. 972–982.
- [134] 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.
- [135] A. KALRA AND D. S. BROWN, *Interpretable reward learning via differentiable decision trees*, in NeurIPS ML Safety Workshop, 2022.
- [136] ———, *Can differentiable decision trees learn interpretable reward functions?*, arXiv preprint arXiv:2306.13004, (2023).
- [137] S. KAMBHAMPATI, S. SREEDHARAN, M. VERMA, Y. ZHA, AND L. GUAN, *Symbols as a lingua franca for bridging human-ai chasm for explainable and advisable ai systems*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 12262–12267.
- [138] M. KENDALL, *Rank Correlation Methods; Griffin, C., Ed*, 1975.
- [139] M. G. KENDALL, *A new measure of rank correlation*, Biometrika, 30 (1938), pp. 81–93.
- [140] E. M. KENNY, M. TUCKER, AND J. SHAH, *Towards interpretable deep reinforcement learning with human-friendly prototypes*, in The Eleventh International Conference on Learning Representations, 2022.
- [141] T. KIM, Y. YUE, S. TAYLOR, AND I. MATTHEWS, *A decision tree framework for spatiotemporal sequence prediction*, in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015, pp. 577–586.
- [142] D. P. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

- [143] W. B. KNOX, A. ALLIEVI, H. BANZHAF, F. SCHMITT, AND P. STONE, *Reward (mis) design for autonomous driving*, Artificial Intelligence, 316 (2023), p. 103829.
- [144] W. B. KNOX, I. R. FASEL, AND P. STONE, *Design principles for creating human-shapable agents.*, in AAI Spring Symposium: Agents that Learn from Human Teachers, 2009, pp. 79–86.
- [145] W. B. KNOX, S. HATGIS-KESSELL, S. BOOTH, S. NIEKUM, P. STONE, AND A. ALLIEVI, *Models of human preference for learning reward functions*, arXiv preprint arXiv:2206.02231, (2022).
- [146] W. B. KNOX AND P. STONE, *Tamer: Training an agent manually via evaluative reinforcement*, in 2008 7th IEEE international conference on development and learning, IEEE, 2008, pp. 292–297.
- [147] P. W. KOH, T. NGUYEN, Y. S. TANG, S. MUSSMANN, E. PIERSON, B. KIM, AND P. LIANG, *Concept bottleneck models*, in International conference on machine learning, PMLR, 2020, pp. 5338–5348.
- [148] R. KOMMIYA MOTHILAL, D. MAHAJAN, C. TAN, AND A. SHARMA, *Towards unifying feature attribution and counterfactual explanations: Different means to the same end*, in Proceedings of the 2021 AAI/ACM Conference on AI, Ethics, and Society, 2021, pp. 652–663.
- [149] A. KOUL, A. FERN, AND S. GREYDANUS, *Learning finite state representations of recurrent policy networks*, in International Conference on Learning Representations, 2019.
- [150] A. KOUL, S. GREYDANUS, AND A. FERN, *Learning finite state representations of recurrent policy networks*, arXiv preprint arXiv:1811.12530, (2018).
- [151] D. LAFOND, S. TREMBLAY, AND S. BANBURY, *Cognitive shadow: A policy capturing tool to support naturalistic decision making*, in 2013 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), IEEE, 2013, pp. 139–142.
- [152] H. LAURENT AND R. L. RIVEST, *Constructing optimal binary decision trees is np-complete*, Information processing letters, 5 (1976), pp. 15–17.
- [153] K. LEE, L. SMITH, A. DRAGAN, AND P. ABBEEL, *B-pref: Benchmarking preference-based reinforcement learning*, Advances in Neural Information Processing Systems, 35 (2021).

- [154] K. LEE, L. M. SMITH, AND P. ABBEEL, *Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training*, in International Conference on Machine Learning, PMLR, 2021, pp. 6152–6163.
- [155] J. LEIKE, D. KRUEGER, T. EVERITT, M. MARTIC, V. MAINI, AND S. LEGG, *Scalable agent alignment via reward modeling: a research direction*, arXiv preprint arXiv:1811.07871, (2018).
- [156] E. LEÓN, *Spaces of convex n -partitions*, PhD thesis, Freie Universität Berlin Berlin, 2015.
- [157] Z.-H. LI, Y. YU, Y. CHEN, K. CHEN, Z. HU, AND C. FAN, *Neural-to-tree policy distillation with policy improvement criterion*, arXiv preprint arXiv:2108.06898, (2021).
- [158] J. LIN, *Divergence measures based on the shannon entropy*, IEEE Transactions on Information theory, 37 (1991), pp. 145–151.
- [159] D. LINDNER, M. TURCHETTA, S. TSCHIATSCHKEK, K. CIOSEK, AND A. KRAUSE, *Information directed reward learning for reinforcement learning*, Advances in Neural Information Processing Systems, 34 (2021), pp. 3850–3862.
- [160] P. LIPTON, *Contrastive explanation*, Royal Institute of Philosophy Supplements, 27 (1990), pp. 247–266.
- [161] Z. C. LIPTON, *The mythos of model interpretability*, Queue, 16 (2018), pp. 31–57.
- [162] M. L. LITTMAN AND C. SZEPESVÁRI, *A generalized reinforcement-learning model: Convergence and applications*, in ICML, vol. 96, 1996, pp. 310–318.
- [163] 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.
- [164] H. LIU, B. KIUMARSI, Y. KARTAL, A. T. KORU, H. MODARES, AND F. L. LEWIS, *Reinforcement learning applications in unmanned vehicle control: A comprehensive overview*, Unmanned Systems, (2022), pp. 1–10.
- [165] Y. LIU, G. DATTA, E. NOVOSELLER, AND D. S. BROWN, *Efficient preference-based reinforcement learning using learned dynamics models*, in NeurIPS workshop on Human in the Loop Learning, 2022.
- [166] S. M. LUNDBERG AND S.-I. LEE, *A unified approach to interpreting model predictions*, Advances in neural information processing systems, 30 (2017).

- [167] S. M. LUNDBERG, B. NAIR, M. S. VAVILALA, M. HORIBE, M. J. EISSES, T. ADAMS, D. E. LISTON, D. K.-W. LOW, S.-F. NEWMAN, J. KIM, ET AL., *Explainable machine-learning predictions for the prevention of hypoxaemia during surgery*, *Nature biomedical engineering*, 2 (2018), pp. 749–760.
- [168] J. LUO, S. GREEN, P. FEGHALI, G. LEGRADY, AND C. K. KOÇ, *Visual diagnostics for deep reinforcement learning policy development*, arXiv preprint arXiv:1809.06781, (2018).
- [169] S. MAHMUD, S. SAISUBRAMANIAN, AND S. ZILBERSTEIN, *Reveale: Reward verification and learning using explanations*, in *AAAI Workshop on Artificial Intelligence Safety*, 2023.
- [170] MATHONLINE, *The Simple Function Approximation Theorem*.
<http://mathonline.wikidot.com/the-simple-function-approximation-theorem>.
- [171] J. MCCALMON, T. LE, S. ALQAHTANI, AND D. LEE, *Caps: Comprehensible abstract policy summaries for explaining reinforcement learning agents*, in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 889–897.
- [172] T. MCGRATH, A. KAPISHNIKOV, N. TOMAŠEV, A. PEARCE, M. WATTENBERG, D. HASABIS, B. KIM, U. PAQUET, AND V. KRAMNIK, *Acquisition of chess knowledge in alp-hazero*, *Proceedings of the National Academy of Sciences*, 119 (2022), p. e2206625119.
- [173] S. MCGREGOR, H. BUCKINGHAM, T. G. DIETTERICH, R. HOUTMAN, C. MONTGOMERY, AND R. METOYER, *Interactive visualization for testing markov decision processes: Mdpvis*, *Journal of Visual Languages & Computing*, 39 (2017), pp. 93–106.
Special Issue on Programming and Modelling Tools.
- [174] E. J. MICHAUD, A. GLEAVE, AND S. RUSSELL, *Understanding learned reward functions*, arXiv preprint arXiv:2012.05862, (2020).
- [175] S. MILANI, N. TOPIN, M. VELOSO, AND F. FANG, *Explainable reinforcement learning: A survey and comparative review*, *ACM Comput. Surv.*, (2023).
- [176] T. MILLER, *Contrastive explanation: A structural-model approach*, arXiv preprint arXiv:1811.03163, (2018).
- [177] ———, *Explanation in artificial intelligence: Insights from the social sciences*, *Artificial Intelligence*, 267 (2019), pp. 1–38.
- [178] B. MILLIDGE, *Towards a mathematical theory of abstraction*, arXiv preprint arXiv:2106.01826, (2021).

- [179] A. MISHRA, U. SONI, J. HUANG, AND C. BRYAN, *Why? why not? when? visual explanations of agent behaviour in reinforcement learning*, in 2022 IEEE 15th Pacific Visualization Symposium (PacificVis), IEEE, 2022, pp. 111–120.
- [180] C. MOLNAR, *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*, 2018.
- [181] E. F. MORALES AND C. SAMMUT, *Learning to fly by combining reinforcement learning with behavioural cloning*, in Proceedings of the twenty-first international conference on Machine learning, 2004, p. 76.
- [182] F. MOSTELLER, *Remarks on the method of paired comparisons: I. the least squares solution assuming equal standard deviations and equal correlations*, Psychometrika, 16 (1951), pp. 3–9.
- [183] R. K. MOTHILAL, A. SHARMA, AND C. TAN, *Explaining machine learning classifiers through diverse counterfactual explanations*, in Proceedings of the 2020 conference on fairness, accountability, and transparency, 2020, pp. 607–617.
- [184] M. MOVIN, G. D. JUNIOR, J. HOLLMÉN, AND P. PAPAPETROU, *Explaining black box reinforcement learning agents through counterfactual policies*, in Advances in Intelligent Data Analysis XXI, Cham, 2023, Springer Nature Switzerland, pp. 314–326.
- [185] S. MURTHY AND S. SALZBERG, *Lookahead and pathology in decision tree induction*, in IJCAI, Citeseer, 1995, pp. 1025–1033.
- [186] S. NAGESHRAO, B. COSTA, AND D. FILEV, *Interpretable approximation of a deep reinforcement learning agent as a set of if-then rules*, in 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), IEEE, 2019, pp. 216–221.
- [187] R. K. NAYYAR, P. VERMA, AND S. SRIVASTAVA, *Differential assessment of black-box ai agents*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 9868–9876.
- [188] A. Y. NG, D. HARADA, AND S. RUSSELL, *Policy invariance under reward transformations: Theory and application to reward shaping*, in Icml, vol. 99, Citeseer, 1999, pp. 278–287.
- [189] A. Y. NG, S. RUSSELL, ET AL., *Algorithms for inverse reinforcement learning.*, in Icml, vol. 1, 2000, p. 2.
- [190] M. L. OLSON, R. KHANNA, L. NEAL, F. LI, AND W.-K. WONG, *Counterfactual state explanations for reinforcement learning agents via generative deep learning*, Artificial Intelligence, 295 (2021), p. 103455.

-
- [191] L. OUYANG, J. WU, X. JIANG, D. ALMEIDA, C. WAINWRIGHT, P. MISHKIN, C. ZHANG, S. AGARWAL, K. SLAMA, A. RAY, ET AL., *Training language models to follow instructions with human feedback*, Advances in Neural Information Processing Systems, 35 (2022), pp. 27730–27744.
- [192] A. PACE, A. CHAN, AND M. VAN DER SCHAAAR, *Poetree: Interpretable policy learning with adaptive decision trees*, in International Conference on Learning Representations, 2022.
- [193] A. PÁEZ, *The pragmatic turn in explainable artificial intelligence (xai)*, Minds and Machines, 29 (2019), pp. 441–459.
- [194] F. PAISCHER, T. ADLER, M. HOFMARCHER, AND S. HOCHREITER, *Semantic helm: An interpretable memory for reinforcement learning*, arXiv preprint arXiv:2306.09312, (2023).
- [195] A. PAN, K. BHATIA, AND J. STEINHARDT, *The effects of reward misspecification: Mapping and mitigating misaligned models*, in International Conference on Learning Representations, 2022.
- [196] T. D. PEREIRA, J. W. SHAEVITZ, AND M. MURTHY, *Quantifying behavior to understand the brain*, Nature neuroscience, 23 (2020), pp. 1537–1549.
- [197] B. D. PIERSON, D. ARENDT, J. MILLER, AND M. E. TAYLOR, *Comparing explanations in RL*, Neural Computing and Applications, (2023), pp. 1–12.
- [198] D. PRUTHI, R. BANSAL, B. DHINGRA, L. B. SOARES, M. COLLINS, Z. C. LIPTON, G. NEUBIG, AND W. W. COHEN, *Evaluating explanations: How much do explanations from the teacher aid students?*, Transactions of the Association for Computational Linguistics, 10 (2022), pp. 359–375.
- [199] E. PUIUTTA AND E. VEITH, *Explainable reinforcement learning: A survey*, in International cross-domain conference for machine learning and knowledge extraction, Springer, 2020, pp. 77–95.
- [200] N. PURI, S. VERMA, P. GUPTA, D. KAYASTHA, S. DESHMUKH, B. KRISHNAMURTHY, AND S. SINGH, *Explain your move: Understanding agent actions using specific and relevant feature attribution*, arXiv preprint arXiv:1912.12191, (2019).
- [201] L. D. PYEATT, *Reinforcement learning with decision trees.*, in 21 st IASTED International Multi-Conference on Applied Informatics, 2003, pp. 26–31.
- [202] J. QIAN, P. WENG, AND C. TAN, *Learning rewards to optimize global performance metrics in deep reinforcement learning*, in Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, 2023, pp. 1951–1960.

- [203] A. RAFFIN, *Rl baselines zoo*.
<https://github.com/araffin/rl-baselines-zoo>, 2018.
- [204] M. RAHTZ, V. VARMA, R. KUMAR, Z. KENTON, S. LEGG, AND J. LEIKE, *Safe deep rl in 3d environments using human feedback*, arXiv preprint arXiv:2201.08102, (2022).
- [205] T. RÄZ, *Ml interpretability: Simple isn't easy*, arXiv preprint arXiv:2211.13617, (2022).
- [206] P. RAZZAGHI, A. TABRIZIAN, W. GUO, S. CHEN, A. TAYE, E. THOMPSON, A. BREGEON, A. BAHERI, AND P. WEI, *A survey on reinforcement learning in aviation applications*, arXiv preprint arXiv:2211.02147, (2022).
- [207] S. REDDY, A. DRAGAN, S. LEVINE, S. LEGG, AND J. LEIKE, *Learning human objectives by evaluating hypothetical behavior*, in International Conference on Machine Learning, PMLR, 2020, pp. 8020–8029.
- [208] M. T. RIBEIRO, S. SINGH, AND C. GUESTRIN, " *why should i trust you?*" *explaining the predictions of any classifier*, in Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 1135–1144.
- [209] M. RIBEIRO FURTADO DE MENDONÇA, A. ZIVIANI, AND A. DA MOTTA SALLES BARRETO, *Abstract state transition graphs for model-based reinforcement learning*, in 2018 7th Brazilian Conference on Intelligent Systems (BRACIS), 2018, pp. 115–120.
- [210] E. H. ROSCH, *Natural categories*, Cognitive psychology, 4 (1973), pp. 328–350.
- [211] S. ROSS, G. GORDON, AND D. BAGNELL, *A reduction of imitation learning and structured prediction to no-regret online learning*, in Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011, pp. 627–635.
- [212] A. M. ROTH, J. LIANG, AND D. MANOCHA, *Xai-n: Sensor-based robot navigation using expert policies and decision trees*, in 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2021, pp. 2053–2060.
- [213] A. M. ROTH, N. TOPIN, P. JAMSHIDI, AND M. VELOSO, *Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy*, arXiv preprint arXiv:1907.01180, (2019).
- [214] C. RUDIN, *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*, Nature Machine Intelligence, 1 (2019), pp. 206–215.
- [215] C. RUPPRECHT, C. IBRAHIM, AND C. J. PAL, *Finding and visualizing weaknesses of deep reinforcement learning agents*, in International Conference on Learning Representations, 2020.

- [216] J. RUSSELL AND E. SANTOS, *Explaining reward functions in Markov decision processes*, in The Thirty-Second International Flairs Conference, 2019.
- [217] S. RUSSELL, *Human compatible: Artificial intelligence and the problem of control*, Penguin, 2019.
- [218] S. RUSSELL AND P. NORVIG, *Artificial Intelligence: A Modern Approach*, Prentice Hall Press, USA, 3rd ed., 2009.
- [219] S. RÜPING, *Learning Interpretable Models*, PhD thesis, TU Dortmund, 2006.
- [220] D. SADIGH, A. D. DRAGAN, S. SASTRY, AND S. A. SESHIA, *Active preference-based learning of reward functions*, in Proceedings of Robotics: Science and Systems (RSS), 2017.
- [221] S. SADRADDINI, S. SHEN, AND O. BASTANI, *Polytopic trees for verification of learning-based controllers*, in Numerical Software Verification, M. Zamani and D. Zufferey, eds., Cham, 2019, Springer International Publishing, pp. 110–127.
- [222] H. B. SAGHEZCHI AND M. ASADPOUR, *Multivariate decision tree function approximation for reinforcement learning*, in International Conference on Neural Information Processing, Springer, 2010, pp. 687–694.
- [223] E. SALDANHA, B. PRAGGASTIS, T. BILLOW, AND D. L. ARENDT, *ReLVis: Visual Analytics for Situational Awareness During Reinforcement Learning Experimentation*, in EuroVis 2019 - Short Papers, J. Johansson, F. Sadlo, and G. E. Marai, eds., The Eurographics Association, 2019.
- [224] L. SAULIÈRES, M. C. COOPER, AND F. D. DE SAINT CYR, *Reinforcement learning explained via reinforcement learning: Towards explainable policies through predictive explanation*, in 15th International Conference on Agents and Artificial Intelligence (ICAART 2023), 2023, pp. à–paraître.
- [225] L. SEMENOVA, C. RUDIN, AND R. PARR, *On the existence of simpler machine learning models*, in Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency, 2022, pp. 1827–1858.
- [226] P. SEQUEIRA AND M. GERVASIO, *Interestingness elements for explainable reinforcement learning: Understanding agents’ capabilities and limitations*, arXiv:1912.09007, (2019).
- [227] L. S. SHAPLEY ET AL., *A value for n-person games*, (1953).
- [228] H. U. SHEIKH, S. KHADKA, S. MIRET, S. MAJUMDAR, AND M. PHIELIPP, *Learning intrinsic symbolic rewards in reinforcement learning*, in 2022 International Joint Conference on Neural Networks (IJCNN), IEEE, 2022, pp. 1–8.

- [229] M. SHVO, T. Q. KLASSEN, AND S. A. MCILRAITH, *Towards the role of theory of mind in explanation*, in Explainable, Transparent Autonomous Agents and Multi-Agent Systems: Second International Workshop, EXTRAAMAS 2020, Auckland, New Zealand, May 9–13, 2020, Revised Selected Papers 2, Springer, 2020, pp. 75–93.
- [230] A. SILVA, M. GOMBOLAY, T. KILLIAN, I. JIMENEZ, AND S.-H. SON, *Optimization methods for interpretable differentiable decision trees applied to reinforcement learning*, in Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics, vol. 108, PMLR, 26–28 Aug 2020, pp. 1855–1865.
- [231] D. SILVER, T. HUBERT, J. SCHRITTWIESER, I. ANTONOGLU, M. LAI, A. GUEZ, M. LANCTOT, L. SIFRE, D. KUMARAN, T. GRAEPEL, ET AL., *A general reinforcement learning algorithm that masters chess, shogi, and go through self-play*, Science, 362 (2018), pp. 1140–1144.
- [232] J. M. V. SKALSE, N. H. R. HOWE, D. KRASHENINNIKOV, AND D. KRUEGER, *Defining and characterizing reward gaming*, in Advances in Neural Information Processing Systems, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, eds., 2022.
- [233] K. SOKOL AND P. FLACH, *One explanation does not fit all: The promise of interactive explanations for machine learning transparency*, KI-Künstliche Intelligenz, 34 (2020), pp. 235–250.
- [234] ———, *Explainability is in the mind of the beholder: Establishing the foundations of explainable artificial intelligence*, arXiv preprint arXiv:2112.14466, (2021).
- [235] K. SOKOL AND P. A. FLACH, *Towards faithful and meaningful interpretable representations*, CoRR, abs/2008.07007 (2020).
- [236] R. J. STERNBERG AND J. A. HORVATH, *Tacit knowledge in professional practice: Researcher and practitioner perspectives*, Psychology Press, 1999.
- [237] R. E. STRAUCH, *“squishy” problems and quantitative methods*, Policy Sciences, 6 (1975), pp. 175–184.
- [238] R. SUKKERD, R. SIMMONS, AND D. GARLAN, *Tradeoff-focused contrastive explanation for MDP planning*, in 2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), IEEE, 2020, pp. 1041–1048.
- [239] R. SUTTON, *The bitter lesson*, Incomplete Ideas (blog), 13 (2019).
- [240] R. S. SUTTON AND A. G. BARTO, *Reinforcement learning: An introduction*, MIT press, 2018.

-
- [241] A. TALATI, D. SADIGH, ET AL., *Aprel: A library for active preference-based reward learning algorithms*, arXiv preprint arXiv:2108.07259, (2021).
- [242] P. TAMBWEKAR, A. SILVA, N. GOPALAN, AND M. GOMBOLAY, *Specifying and interpreting reinforcement learning policies through simulatable machine learning*, arXiv preprint arXiv:2101.07140, (2021).
- [243] C. TANG AND Y.-C. LAI, *Deep reinforcement learning automatic landing control of fixed-wing aircraft using deep deterministic policy gradient*, in 2020 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, 2020, pp. 1–9.
- [244] M. E. TAYLOR, *Reinforcement learning requires human-in-the-loop framing and approaches*.
- [245] S. TESO AND K. KERSTING, *Explanatory interactive machine learning*, in Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society, 2019, pp. 239–245.
- [246] L. L. THURSTONE, *A law of comparative judgment.*, Psychological review, 34 (1927), p. 273.
- [247] J. TIEN, J. Z.-Y. HE, Z. ERICKSON, A. DRAGAN, AND D. S. BROWN, *Causal confusion and reward misidentification in preference-based reward learning*, in The Eleventh International Conference on Learning Representations, 2023.
- [248] N. TINBERGEN, *On aims and methods of ethology*, Zeitschrift für tierpsychologie, 20 (1963), pp. 410–433.
- [249] N. TOPIN AND M. VELOSO, *Generation of policy-level explanations for reinforcement learning*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 2514–2521.
- [250] S. TSIRTISIS, A. DE, AND M. G. RODRIGUEZ, *Counterfactual explanations in sequential decision making under uncertainty*, in Advances in Neural Information Processing Systems, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds., 2021.
- [251] O. TURNBULL, J. LAWRY, M. LOWENBERG, AND A. RICHARDS, *A cloned linguistic decision tree controller for real-time path planning in hostile environments*, Fuzzy Sets and Systems, 293 (2016), pp. 1–29.
- [252] W. T. UTHER AND M. M. VELOSO, *Tree based discretization for continuous state space reinforcement learning*, in AAAI/IAAI, 1998, pp. 769–774.
- [253] J. VAN DER WAA, J. VAN DIGGELEN, K. V. D. BOSCH, AND M. NEERINCX, *Contrastive explanations for reinforcement learning in terms of expected consequences*, in IJCAI/ECAI Workshop on Explainable Artificial Intelligence, 2018.

- [254] B. VAN FRAASSEN, *The pragmatic theory of explanation*, Theories of Explanation, 8 (1988), pp. 135–155.
- [255] J. VAN OIJEN, G. POPPINGA, O. BROUWER, A. ALIKO, AND J. J. ROESSINGH, *Towards modeling the learning process of aviators using deep reinforcement learning*, in 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2017, pp. 3439–3444.
- [256] M. VASIC, A. PETROVIC, K. WANG, M. NIKOLIC, R. SINGH, AND S. KHURSHID, *Moët: Interpretable and verifiable reinforcement learning via mixture of expert trees*, arXiv preprint arXiv:1906.06717, (2019).
- [257] J. R. VÁZQUEZ-CANTELI AND Z. NAGY, *Reinforcement learning for demand response: A review of algorithms and modeling techniques*, Applied energy, 235 (2019), pp. 1072–1089.
- [258] M. VELOSO, T. BALCH, D. BORRAJO, P. REDDY, AND S. SHAH, *Artificial intelligence research in finance: discussion and examples*, Oxford Review of Economic Policy, 37 (2021), pp. 564–584.
- [259] K. V. VEMURU, S. D. HARBOUR, AND J. D. CLARK, *Reinforcement learning in aviation, either unmanned or manned, with an injection of ai*, in 20th International Symposium on Aviation Psychology, 2019, p. 492.
- [260] A. VERMA, V. MURALI, R. SINGH, P. KOHLI, AND S. CHAUDHURI, *Programmatically interpretable reinforcement learning*, in International Conference on Machine Learning, PMLR, 2018, pp. 5045–5054.
- [261] S. WACHTER, B. MITTELSTADT, AND C. RUSSELL, *Counterfactual explanations without opening the black box: Automated decisions and the GDPR*, Harv. JL & Tech., 31 (2017), p. 841.
- [262] J. WANG, L. GOU, H.-W. SHEN, AND H. YANG, *Dqnviz: A visual analytics approach to understand deep q-networks*, IEEE transactions on visualization and computer graphics, 25 (2018), pp. 288–298.
- [263] N. WANG, D. V. PYNADATH, AND S. G. HILL, *The impact of pomdp-generated explanations on trust and performance in human-robot teams*, in Proceedings of the 2016 international conference on autonomous agents & multiagent systems, 2016, pp. 997–1005.
- [264] N. WILDE, A. BLIDARU, S. L. SMITH, AND D. KULIĆ, *Improving user specifications for robot behavior through active preference learning: Framework and evaluation*, The International Journal of Robotics Research, 39 (2020), pp. 651–667.

- [265] C. WIRTH, R. AKROUR, G. NEUMANN, J. FÜRNKRANZ, ET AL., *A survey of preference-based reinforcement learning methods*, Journal of Machine Learning Research, 18 (2017), pp. 1–46.
- [266] C. WIRTH, J. FÜRNKRANZ, AND G. NEUMANN, *Model-free preference-based reinforcement learning*, in Thirtieth AAAI Conference on Artificial Intelligence, 2016.
- [267] Z. YANG, S. BAI, L. ZHANG, AND P. H. TORR, *Learn to interpret atari agents*, arXiv preprint arXiv:1812.11276, (2018).
- [268] D. YARATS, R. FERGUS, A. LAZARIC, AND L. PINTO, *Reinforcement learning with prototypical representations*, in International Conference on Machine Learning, PMLR, 2021, pp. 11920–11931.
- [269] 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, 33 (2020), pp. 18375–18386.
- [270] Y. YILDIZ, A. AGOGINO, AND G. BRAT, *Predicting pilot behavior in medium-scale scenarios using game theory and reinforcement learning*, Journal of Guidance, Control, and Dynamics, 37 (2014), pp. 1335–1343.
- [271] L. A. ZADEH, *Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic*, Fuzzy sets and systems, 90 (1997), pp. 111–127.
- [272] 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.
- [273] G. ZHANG AND H. KASHIMA, *Learning state importance for preference-based reinforcement learning*, Machine Learning, (2023), pp. 1–17.
- [274] Y. ZHANG, H. WU, AND L. CHENG, *Some new deformation formulas about variance and covariance*, in 2012 Proceedings of International Conference on Modelling, Identification and Control, IEEE, 2012, pp. 987–992.
- [275] Z.-H. ZHOU, *Rule extraction: Using neural networks or for neural networks?*, Journal of Computer Science and Technology, 19 (2004), pp. 249–253.

