TRIPLETREE: A Versatile Interpretable Representation of Black Box Agents and their Environments

Tom Bewley and Jonathan Lawry
Department of Engineering Mathematics, University of Bristol, Bristol, United Kingdom
{tom.bewley, j.lawry}@bristol.ac.uk

Abstract

In explainable artificial intelligence, there is increasing interest in understanding the behaviour of autonomous agents to build trust and validate performance. Modern agent architectures, such as those trained by deep reinforcement learning, are currently so lacking in interpretable structure as to effectively be black boxes, but insights may still be gained from an external, behaviourist perspective. Inspired by conceptual spaces theory, we suggest that a versatile first step towards general understanding is to discretise the state space into convex regions, jointly capturing similarities over the agent's action, value function and temporal dynamics within a dataset of observations. We create such a representation using a novel variant of the CART decision tree algorithm, and demonstrate how it facilitates practical understanding of black box agents through prediction, visualisation and rule-based explanation.

Introduction

This paper explores representational tools for understanding the behaviour of extant autonomous agents while treating them and their environments as black boxes. In popular taxonomies of explainable artificial intelligence (XAI), this is categorised as post hoc, model-agnostic, global explanation (Adadi and Berrada 2018). While black box behaviourist analysis is inherently limited (Chomsky 1959), it can nonetheless serve the practical goals of XAI, which include building trust among human stakeholders and validating the performance of safety-critical systems. It is also viable in contexts where theoretical understanding of the agent's internal mechanism is lacking, as with modern deep learning systems (Samek, Wiegand, and Müller 2017), or where access to this mechanism is impractical or restricted.

We introduce a new data-driven model of black box agents, called TRIPLETREE, which builds on the flexible and interpretable architecture of a binary decision tree. As such, it provides a powerful tool for answering many meaningful questions about agent behaviour.

Problem Setup

Suppose that we need to understand the behaviour, performance and possible failure modes of an autonomous agent operating within a dynamic and complex environment. A priori, we know nothing of the agent's provenance – its governing policy may be a product of reinforcement learning (RL), optimal control algorithms, evolution, or explicit manual design – but we assume that it can be analysed using the theoretical formalism of a *Markov decision process* (MDP). We refer readers to (Sutton and Barto 2018) for an overview of MDPs, and adopt the MDPNv1 notational standard (Thomas and Okal 2015).

To learn anything about the agent we must gather data, and in doing so we make the *black box assumption* (Wachter, Mittelstadt, and Russell 2017; Guidotti et al. 2019; Coppens et al. 2019). That is, we take the role of an observer of the agent-environment complex, with no access to the internal structure of either system, but the ability to record environment states $s \in \mathcal{S}$, agent actions $a \in \mathcal{A}$, and instantaneous rewards $r \in \mathcal{R}$, and their order of occurrence. We thereby assemble $\mathcal{D}$, an ordered dataset of triplets (S_t, A_t, R_t). S_t, A_t and R_t are the state, action and reward from one timestep, uniquely indexed by t. If the MDP is episodic (see (Sutton and Barto 2018)), we keep a record of which states are the initial and terminal ones in each episode. Importantly, we assume that states are represented by vectors of real-valued features, each with a straightforward semantic interpretation (such as a physical quantity). In doing so, we bypass a challenging phase of state representation learning.[1]

The data in $\mathcal{D}$ are generated by the interaction of opaque and complex mechanisms. How might they get us to a position of understanding? Following prior work across the academic spectrum (Carnap 1967; Rosch et al. 1976; Edelman 1998), we take the view that understanding arises by searching for similarities in observed data. A variant of this idea is Gärdenfors' theory of *conceptual spaces* (Gärdenfors 2004).

Conceptual Spaces and Decision Trees

Gärdenfors views sensory observations as embedded in high-dimensional mathematical spaces, and proposes that the building blocks of abstract reasoning are convex regions of such spaces, within which all contained observations are similar according to some salient measure. Such regions are deemed *natural properties* of the system being observed, and can be combined to form semantic concepts such as ob-

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1] We address the problem of interpretable feature construction and selection in (Bewley, Lawry, and Richards 2020).

jects, categories, actions and events. For our purposes, the observations $\mathcal{D}$ do indeed lie within a mathematical space: the MDP state space $\mathcal{S}$. We consider how these observations can be grouped into convex regions of $\mathcal{S}$ based on context-specific measures of similarity. Rather than talking about the system on a state-by-state basis, we may then analyse such regions as meaningful entities in themselves, within which the agent behaves in predictable ways, and between which it moves in predictable patterns. This is a kind of MDP *state abstraction*, informed by observations of the agent itself.

A central issue is choosing which attributes to use for measuring similarity within regions. An obvious criterion is the agent's action. If the agent takes the same action throughout a significant region of $\mathcal{S}$, then it seems that region is worthy of being explicitly represented. Alternatively, we might be interested in the agent's performance as measured by the reward function, and thus measure similarity using the reward elements in $\mathcal{D}$. In practice, it is likely more informative to invoke the notion of value, which is the expected sum of reward after the agent visits each state, temporally discounted by $\gamma \in [0, 1]$. An empirical value estimate can be computed for each sample $t \in \mathcal{D}$ using the rewards of successive samples: $V_t = \sum_{k=0}^{T} \gamma^k R_{t+k}$. Here, T is the time until termination in an episodic MDP, and ∞ otherwise. A third valid option is to define similarity via the temporal dynamics of the MDP itself. These can be neatly captured by the time derivatives of state features, which in discrete time systems are equal to the change in state between timesteps. For each sample $t \in \mathcal{D}$, we define this as $S_{t+1} - S_t$.

Our key assertion in this paper is that there is no need to choose between these three sources of similarity, and that a powerful and versatile model results from identifying regions of $\mathcal{S}$ that are similar from *all three* perspectives. We complete our model by calculating transition probabilities between regions – the probability that having being observed at a state in one region, the agent will move to another region next. These effectively define a probabilistic finite state machine (FSM) model of the agent-environment dynamics, and can be estimated by harnessing the temporal ordering of $\mathcal{D}$.

Figure 1 summarises the proposed model. As we hope to demonstrate, it enables us to make sense of pertinent questions about the key invariances and changes points in the agent's behaviour, the environmental factors responsible for this behaviour and perturbations which would alter it, the regions of $\mathcal{S}$ most commonly visited, and the most likely trajectories between particular states of interest.

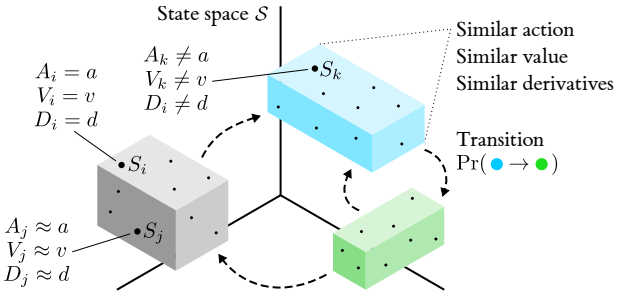


Figure 1: A general model of agent-environment dynamics.

We now turn to the question of practical implementation. There exists an ideal computational tool for finding internally-similar convex regions of mathematical spaces: the humble *decision tree*. A decision tree is 'grown' to predict an output label by recursive binary partitioning of a space of input features. Typically, partitions are axis-aligned, so each resultant region (corresponding to a leaf of the tree) is a hyperrectangle. Partitions are chosen to greedily minimise a measure of impurity in the output labels of training data at each node, which is tantamount to finding and preserving similarity as conceived in our conceptual space model. Trees are popular in XAI, and often hailed as the gold standard of interpretability (Wan et al. 2020). Their hierarchical structure means that global functionality can be analysed as the concatenation of local effects, without higher-order interaction. Their alternative representation as rule sets in disjunctive normal form also enables factual and counterfactual explanation of their outputs (Guidotti et al. 2019).

The sub-field of agent explainability has seen enthusiastic uptake of decision trees (Bastani, Pu, and Solar-Lezama 2018; Coppens et al. 2019; Bewley, Lawry, and Richards 2020). However, these works focus exclusively on predicting and explaining a black box agent's single next action in any given state, effectively approximating its policy function, denoted by π. This approach uses only the first two elements of the triplets (S_t, A_t, R_t) in $\mathcal{D}$, and ignores their ordering, thus foregoes an opportunity for far richer analysis. The all-important notion of value, as well as the temporal dynamics of agent-environment interaction, are entirely absent from a policy-only model, which in isolation is insufficient for answering many reasonable questions about the target agent's performance and dynamical properties.

This lack of versatility is a product of how data are stored in the tree, and of the algorithm used to grow it, rather than an inherent limitation of the decision tree paradigm. We propose to extend the standard notion of impurity to capture multiple facets agent-environment interaction, making full use of the data in $\mathcal{D}$, and providing a practical implementation of the conceptual model in figure 1. Concretely, we grow the tree using a hybrid of three impurity measures related to the agent's action, expected value and state feature time derivatives, hence our new model's name: TRIPLETREE. By modifying a weight vector $\theta \in \mathbb{R}^3_+$, which sets the influence of the three measures, we can smoothly trade off between three types of interpretable model of the system:

- $\theta = [1, 0, 0]$: A conventional policy-only model.
- $\theta = [0, 1, 0]$: A value function.
- $\theta = [0, 0, 1]$: A model of the environment state dynamics.

Any other weighting gives a blended combination of the three, allowing for multifactorial analysis.

The TRIPLETREE Model

Basic Structure: CART

TRIPLETREE is an extension of the CART algorithm (Breiman et al. 1984), which we briefly introduce first. We assume that given a dataset $\mathcal{D}$, CART is being used to predict the agent's action given the state; a policy-only model.

Let $\mathcal{I} = \{1, ..., |\mathcal{D}|\}$ be the set of timestep indices in $\mathcal{D}$, used to initialise the tree's root node before any partitions are made. Let $\mathcal{I}_N$ be the subset of $\mathcal{I}$ at any given node N. To split this node in two, thereby growing the tree, CART searches over binary partitions $\mathcal{I}_N = \{\mathcal{I}_0, \mathcal{I}_1\}$ such that for some state feature f and numerical threshold $\tau \in \mathbb{R}$:

$$(\forall t \in \mathcal{I}_0 \; S_t^{(f)} < \tau) \; \wedge \; (\forall t' \in \mathcal{I}_1 \; S_{t'}^{(f)} \geq \tau). \quad (1)$$

Here, $S_t^{(f)}$ is the fth element of the state vector S_t. For each candidate partition, we calculate the population-weighted reduction in a measure of action label impurity I induced by dividing the set into these two parts. For a discrete action space $\mathcal{A}$, the Gini impurity is used:

$$\mathrm{I}_N = \frac{1}{|\mathcal{I}_N|^2} \sum_{a \in \mathcal{A}} \mathrm{count}(\mathcal{I}_N, a)(1 - \mathrm{count}(\mathcal{I}_N, a)), \quad (2)$$

where $\mathrm{count}(\mathcal{I}_N, a) = |\{t \in \mathcal{I}_N : A_t = a\}|$. For continuous actions $\mathcal{A} = \mathbb{R}$, the impurity measure is the variance:

$$\mathrm{I}_N = \frac{1}{2|\mathcal{I}_N|^2} \sum_{t \in \mathcal{I}_N} \sum_{t' \in \mathcal{I}_N} (A_t - A_{t'})^2. \quad (3)$$

The quality Q of a candidate partition is defined as:

$$\mathrm{Q}(\mathcal{I}_N, \{\mathcal{I}_0, \mathcal{I}_1\}) = \mathrm{I}_N - \frac{\mathrm{I}_0|\mathcal{I}_0| + \mathrm{I}_1|\mathcal{I}_1|}{|\mathcal{I}_N|}. \quad (4)$$

CART selects the partition that maximises Q, and the corresponding feature f and threshold τ are recorded in the tree. $\mathcal{I}_0$ and $\mathcal{I}_1$ also define the members of two new child nodes of N. CART proceeds to search for the best binary partition of each child, and grows the tree depth-first up to a stopping condition, such as a depth limit.

In any tree, a subset of nodes, called the *leaves* $\mathcal{L}$, remain childless. Every sample in $\mathcal{D}$ is a member of exactly one leaf L, whose key attribute is an action prediction $\tilde{a}_L \in \mathcal{A}$. For discrete $\mathcal{A}$, this is typically the modal action among the leaf's constituent samples: $\tilde{a}_L = \mathrm{argmax}_a \, \mathrm{count}(\mathcal{I}_L, a)$. For continuous $\mathcal{A}$, the mean is used: $\tilde{a}_L = \sum_{t \in \mathcal{I}_L} A_t / |\mathcal{I}_L|$. To predict an action for an unseen state vector s, the tree propagates s down a path from the root by comparing its features to the thresholds encountered, then returns the prediction of the leaf which is ultimately reached.

Modifications and Extensions

Our fundamental divergence from CART is in the use of the entire content and strucure of $\mathcal{D}$. TRIPLETREE accepts ordered triplet samples of the form (S_t, A_t, R_t), and prior to commencing growth evaluates two additional attributes, namely the value estimate $V_t = \sum_{k=0}^{T} \gamma^k R_{t+k}$ and state derivative vector $D_t = S_{t+1} - S_t$. Rather than using only the agent's action as an output label, each leaf L is associated with three predictions: the action $\tilde{a}_L$, a value estimate $\tilde{v}_L$ (the mean of the leaf's contituent samples), and a state derivative estimate $\tilde{d}_L$ (the elementwise mean).

The TRIPLETREE growth algorithm trades off the ability to make these three kinds of prediction by encouraging leaves to have low variability in all three attributes across their constituent samples. To achieve this, we compute three measures of the quality of candidate partitions:

- **Action quality** $\mathrm{Q}^{(A)}$: defined exactly as in equation 4.
- **Value quality** $\mathrm{Q}^{(V)}$: defined equivalently, but using the variance in value estimates as the impurity measure $\mathrm{I}^{(V)}$.
- **Derivative quality** $\mathrm{Q}^{(D)}$: defined equivalently, but using an impurity measure $\mathrm{I}^{(D)}$ that sums the variance in derivatives across all d of the feature dimensions:

$$\mathrm{I}_N^{(D)} = \frac{1}{2|\mathcal{I}_N|^2} \sum_{f=1}^{d} \frac{1}{\sigma^{(f)}} \sum_{t \in \mathcal{I}_N} \sum_{t' \in \mathcal{I}_N} (D_t^{(f)} - D_{t'}^{(f)})^2. \quad (5)$$

  $1/\sigma^{(f)}$ is a normalisation factor for each derivative – the reciprocal of its standard deviation across $\mathcal{D}$ – which prevents features with large magnitudes dominating the impurity calculation. This sum-of-variances impurity measure is similar to those used in prior work on multivariate regression trees (De'Ath 2002; Kim et al. 2015).

After computing $\mathrm{Q}^{(A)}$, $\mathrm{Q}^{(V)}$ and $\mathrm{Q}^{(D)}$, we aggregate them into a hybrid measure of partition quality Q^*. Having experimented with alternative methods in various MDP contexts, we find that a linear combination provides a good compromise of simplicity, robustness and flexibility:

$$\mathrm{Q}^* = \left[\frac{\mathrm{Q}_N^{(A)}}{\mathrm{I}_{\text{root}}^{(A)}}, \frac{\mathrm{Q}_N^{(V)}}{\mathrm{I}_{\text{root}}^{(V)}}, \frac{\mathrm{Q}_N^{(D)}}{\mathrm{I}_{\text{root}}^{(D)}} \right] \cdot \theta. \quad (6)$$

Here we have omitted the arguments of the quality terms for brevity. $\theta \in \mathbb{R}_+^3$ 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 at the root node (i.e. before any partitions are made). This brings the three measures onto equivalent scales.

CART follows a depth-first growth strategy, which is known to lead to suboptimal allocation of partitions. We depart from this by adopting a simple best-first strategy for selecting which leaf node to partition at each stage of tree growth. Again taking a hybrid view of impurity, we identify the best current leaf to partition, L_{best}, as follows:

$$L_{\text{best}} = \underset{L \in \mathcal{L}}{\mathrm{argmax}} \, |\mathcal{I}_L| \left[\frac{\mathrm{I}_L^{(A)}}{\mathrm{I}_{\text{root}}^{(A)}}, \frac{\mathrm{I}_L^{(V)}}{\mathrm{I}_{\text{root}}^{(V)}}, \frac{\mathrm{I}_L^{(D)}}{\mathrm{I}_{\text{root}}^{(D)}} \right] \cdot \theta, \quad (7)$$

where θ is the same as in equation (6). This approach prioritises the partitioning of leaves with high total impurity, weighted by their sample counts. Our criterion for terminating tree growth is a limit on the number of leaves, $|\mathcal{L}|$.

The final feature of TRIPLETREE is the calculation of leaf-to-leaf transition probabilities. Let $\mathrm{leaf}(t) = L \in \mathcal{L} : t \in \mathcal{I}_L$ be the leaf at which a sample t resides. For terminal samples in episodic MDPs, we define $\mathrm{leaf}(t) = \emptyset$. Furthermore, let $\mathcal{I}_L^* = \{t \in \mathcal{I}_L : \mathrm{leaf}(t-1) \neq L\}$ be the subset of $\mathcal{I}_L$ whose predecessors are not themselves in $\mathcal{I}_L$: the first in each sequence of successive observations that reside at L. We perform our calculations at the level of sequences, rather than individual samples, to avoid a double-counting effect. For each sequence-starting sample $t \in \mathcal{I}_L^*$, we find the length of its successor sequence, $\mathrm{seqlen}(t)$, and the leaf containing the sample that breaks it, $\mathrm{nextleaf}(t)$:

$$\begin{gathered} \mathrm{seqlen}(t) = \min\{k : \mathrm{leaf}(t+k) \neq \mathrm{leaf}(t)\} \; ; \\ \mathrm{nextleaf}(t) = \mathrm{leaf}(t + \mathrm{seqlen}(t)). \end{gathered} \quad (8)$$

For a given source leaf L and destination leaf L', we are interested in the subset of $\mathcal{I}^*_L$ whose successor sequences are followed by a transition to L':

$$\mathcal{I}^*_{L\to L'} = \{t \in \mathcal{I}^*_L : \text{nextleaf}(t) = L'\}. \quad (9)$$

Note that in episodic MDPs, $\mathcal{I}^*_{L\to\emptyset}$ is well-defined and meaningful; it contains the members of $\mathcal{I}^*_L$ for whom the episode terminates before a transition to another leaf. We can now compute the empirical probability that any given sequence in L ends in a transition to L', and the mean length of such a sequence:

$$P_L(L') = \frac{|\mathcal{I}^*_{L\to L'}|}{|\mathcal{I}^*_L|} \; ; \; T_L(L') = \sum_{t\in\mathcal{I}^*_{L\to L'}} \frac{\text{seqlen}(t)}{|\mathcal{I}^*_{L\to L'}|}. \quad (10)$$

Transition probabilities and times are stored at their respective source leaves (L here) as further attributes alongside the predictions $\tilde{a}_L$, $\tilde{v}_L$ and $\tilde{d}_L$.

In summary, the key features of TRIPLETREE are:

- Acceptance of the triplet observations in $\mathcal{D}$, calculation of value and state derivatives for each sample, and storage of predicted values of these variables at each leaf.
- A hybrid measure of partition 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_L and T_L to encode information about temporal dynamics in terms of leaf-to-leaf transitions.
- A best-first growth strategy.

A Python implementation of TRIPLETREE is available on GitHub at https://github.com/tombewley/TripleTree.

Related Work

Before the widespread adoption of deep neural network function approximators in RL, decision trees were used to create discretised state abstractions for tabular Q-learning algorithms (Uther and Veloso 1998). Tree models have also been used to learn a value function with the aim of creating an interpretable agent that performs well in the task environment (Pyeatt 2003; Roth et al. 2019), and also to mimic the value function of an existing black box policy as a route to explainability (Liu et al. 2018). This latter model also keeps track of transition probabilities between tree leaves, similarly to our approach. In (Jiang, Hwang, and Lin 2019), a decision tree is grown to minimise the impurity of environment state derivatives as part of a model-based RL framework, and in (Kim et al. 2015) 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 machine for visualisation and analysis (Koul, Fern, and Greydanus 2018).

We know of one work that considers a hybrid action- and value-based tree impurity measure (Saghezchi and Asadpour 2010), but the idea is tangential to the main topic of the paper and its implications left unconsidered. We are unaware of any prior work that jointly represents the policy, value function and temporal dynamics in one decision tree, or considers the benefits of doing so for interpretability.

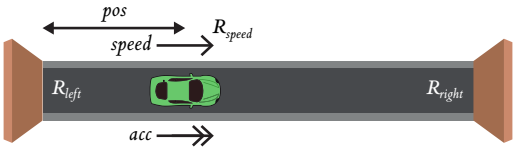


Figure 2: The 2-dimensional road MDP.

Prediction Tradeoff Experiments

We initially validate TRIPLETREE in a simple MDP with 2 state features and 2 discrete actions. This can be interpreted as a straight road, down which a vehicle agent can drive in either direction. The state features are position $pos \in [0, 3]$ (increasing left-to-right) and speed $speed \in [-0.1, 0.1]$, and the agent's action is a small positive or negative acceleration $acc \in \{-0.001, 0.001\}$. 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: $R_{speed} \times |speed|$. Figure 2 summarises this information.

For a given R_{left}, R_{right}, R_{speed}, discount factor γ (we use $\gamma = 0.99$), and suitable discretisation of $\mathcal{S}$ (we use a 30×30 grid) an optimal policy can be found by dynamic programming (DP). We use the DP policies for four reward function variants 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.

Figure 3 shows the result of growing a TRIPLETREE of up to 200 leaves using these four datasets, with various impurity weightings θ. The columns show predictive losses for action (proportion of incorrect predictions), value (RMS error) and derivatives (dot product of RMS error with normalisation factors $1/\sigma$) as a function of leaf count. 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 partition 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 indicates that in this MDP, there tend to be regions of $\mathcal{S}$ in which value varies significantly but the agent's action and state derivatives do not, thereby creating a conflict as to which leaves are worthy of partitioning. This phenomenon is most pronounced for the policy on the bottom row. Another notable trend is that partitioning on derivatives alone does very well in terms of action loss. This makes perfect sense once we realise that the agent's action (acc) is exactly the time derivative of one of the state features ($speed$).

This analysis begs the question: what is the best θ for this MDP? Ultimately, the answer depends on the intended application, but if general versatility is important then we may wish to minimise the *worst* of the three loss types. Our analysis along these lines (see Appendix A) suggests that in this MDP, a good compromise is attained by placing increased weight on value impurity: $\theta = [0.2, 0.6, 0.2]$.

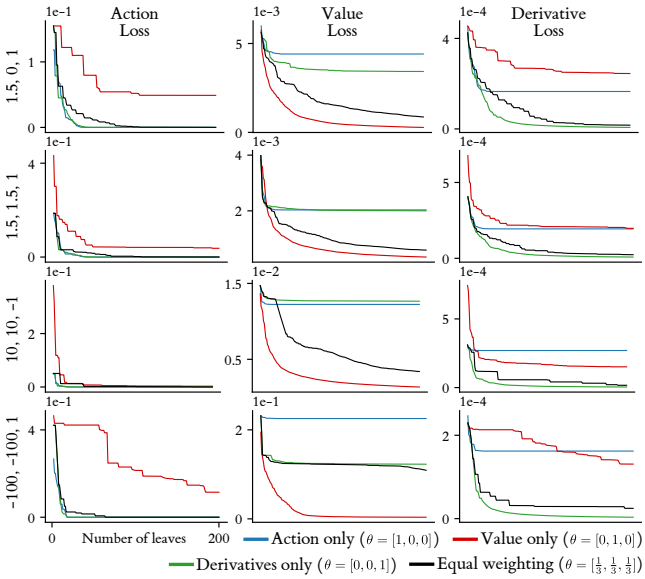


Figure 3: Prediction losses for four variants of the road MDP, with R_{left}, R_{right}, R_{speed} as stated in the left-hand labels.

Multiattribute Visualisation in State Space

Recall that in a decision tree, each leaf is associated with a hyperrectangle in the d-dimensional state space $\mathcal{S}$, whose boundaries correspond to the partitions of its ancestor nodes. If $d \in \{1, 2\}$, hyperrectangles reduce to lines or rectangles, which can be directly shown on axes corresponding to $\mathcal{S}$ itself (we return to the $d > 2$ case later in this paper). Each leaf can be coloured according to some salient attribute including, but not limited to, one of its three predictions.

Figure 4 demonstrates the rich information conveyed by such visualisations in the road MDP. Each row of plots is generated from a single TRIPLETREE with 200 leaves, grown using the compromise weighting $\theta = [0.2, 0.6, 0.2]$. In the first column, leaves are coloured by predicted action, revealing the optimal DP policies. The decision boundaries 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 feature in 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 components. 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, we show it as a quiver plot 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. Quiver plots provide an excellent high-level overview

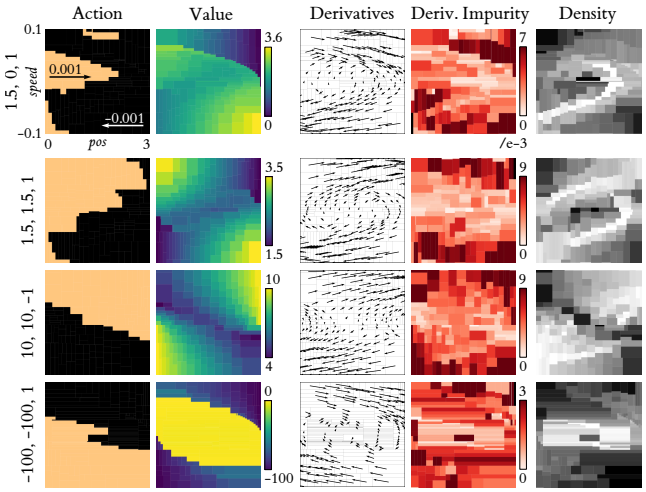


Figure 4: Five types of visualisation using TRIPLETREE.

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 $\mathcal{S}$ we should be most confident in the model's derivative predictions. We can also create equivalent plots for action and value impurity. The final colouring attribute is sample density, computed by dividing the population of each leaf by its volume in $\mathcal{S}$: 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.

Rule-based Explanation

A popular interpretability feature of decision trees is the generation of textual explanations of outputs in terms of the decision rules applied. The simplest type of rule-based explanation is a *factual* one. For any leaf $L \in \mathcal{L}$, simply enumerating the boundaries of the leaf's hyperrectangle describes the region of $\mathcal{S}$ within which a constant prediction holds. Figure 5a shows a portion of the action visualisation for one of the 200-leaf trees from the previous section. The action for state S_t can be explained factually as follows:

"Action $= 0.001$ because $pos \in [1.1, 1.32]$ and $speed \in [0.021, 0.045]$."

It is argued that a more natural (Lipton 1990) and legally persuasive (Wachter, Mittelstadt, and Russell 2017) form of explanation is the *counterfactual*, which provides reasons why an alternative outcome, known as a foil, does not occur instead. In our context, the foil is an action other than the one taken in S_t. After enumerating all leaves which predict the foil action, we choose one, then find the change in state required to move to a location s' in that leaf. There is much debate about how to select s' from many alternatives (Guidotti et al. 2019; Poyiadzi et al. 2020), which often hinges on the notion of a *minimal* change in state. In TRIPLETREE, we use a two-stage process based on the L_0 and L_2 norms (Appendix B), which gives the following minimal counterfactual for the action in figure 5a:

"Action would $= -0.001$ if $speed \geq 0.045$."

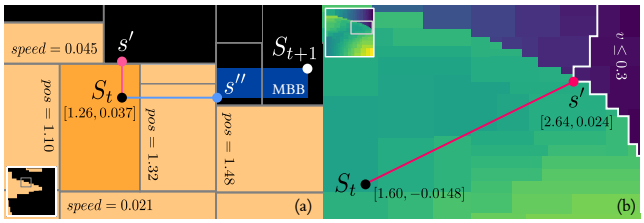


Figure 5: Various forms of explanation for action and value.

For the ordered data in $\mathcal{D}$, a third form of explanation is *temporal*, which explains changes over time such as the action change from S_t to S_{t+1} in figure 5a. This again takes a counterfactual perspective, although a subtlety is that using S_{t+1} directly as a foil does not produce a minimal explanation. Our method for resolving this (also in Appendix B) finds s'', the minimal foil from S_t subject to the constraint that the minimum bounding box (MBB) of s'' and S_{t+1} only intersects leaves with the same action as S_{t+1}. This yields:

"Action changed $0.001 \rightarrow -0.001$ because $pos \geq 1.48$."

Temporal explanation could be extended to a longer sequence of samples by identifying all timesteps at which the action changes, explaining each as above, and combining them into a behavioural story using the conjunction "then".

The TRIPLETREE model allows us to similarly explain value predictions $\tilde{v}_L$. Since value is continuous, a counterfactual could explain why value is less than or greater than a threshold, rather than defining a precise numerical foil which will only ever be made by one leaf at most. In figure 5b, we can see that the foil condition $v \leq 0.3$ leads to the following minimal counterfactual for the value at S_t:

"Value would ≤ 0.3 if $pos \geq 2.64$ and $speed \geq 0.024$."

Trajectory Simulation

For each leaf L, the derivative prediction $\tilde{d}_L$ and transition probabilities P_L both describe the agent's movement through a region of $\mathcal{S}$. These can be combined to construct behavioural trajectories which may never have occurred in $\mathcal{D}$, but are nonetheless realistic given the agent's policy. This could be useful for answering targeted queries about how it navigates before, after and between states of interest.

Here we consider the problem of finding such a trajectory between a given start leaf L_S and an end leaf L_E. We start by using Dijkstra's algorithm (Dijkstra 1959) to find a sequence of leaves $\mathcal{L}_{S\rightarrow E} = (L_S, L_1, L_2, ..., L_E)$ that the agent moves through with nonzero probability. We define the (inverse) cost of each transition $L \rightarrow L'$ according to the probability $P_L(L')$, and the cost of a full sequence as the product of its constituent transitions. If valid sequences exist between L_S and L_E, Dijkstra's algorithm is guaranteed to find the highest-probability one first. If no solutions exist the algorithm returns a null result, which still gives valuable information about the non-reachability of states.

To generate a realistic trajectory through $\mathcal{L}_{S\rightarrow E}$, we solve a constrained optimisation problem to build a piecewise linear path whose segments are well aligned with the leaves' predicted derivative vectors. Concretely, for each leaf $L_j \in$

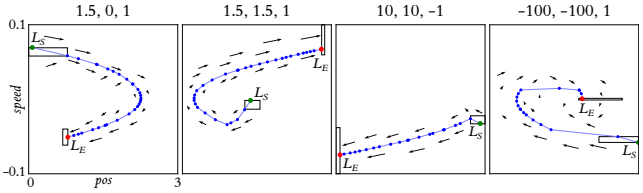


Figure 6: Simulated trajectories in the road MDP.

$\mathcal{L}_{S\rightarrow E}$ we initialise a path node p_j on the hyperrectangle boundary, then perform gradient descent updates on all node locations to minimise the squared angular deviation between the path segments and the derivative vectors. When calculating angles, we normalise derivatives by the vector of inverse standard deviations across $\mathcal{D}$, $1/\sigma$. The unconstrained update to p_j is proportional to the partial derivative of the squared deviations for the segments before and after:

$$\frac{\partial}{\partial p_j}\left[\left(\cos^{-1}\frac{x_j \cdot d_j}{||x_j||||d_j||}\right)^2 + \left(\cos^{-1}\frac{x_{j+1} \cdot d_{j+1}}{||x_{j+1}||||d_{j+1}||}\right)^2\right], \tag{11}$$

where $x_j = (p_j - p_{j-1}) \circ 1/\sigma$ and $d_j = \tilde{d}_{L_j} \circ 1/\sigma$ ($\circ$ denotes the Hadamard product). Rather than applying the update directly, we constrain the node to remain on its respective boundary, and always 'visible' from the previous node (i.e. it never moves to the far side of the boundary).

Figure 6 contains trajectories generated by this search-then-align method from the 200-leaf TRIPLETREES. Derivative arrows from nearby leaves indicate that the trajectories align well with agents' true motion in each region of $\mathcal{S}$. The optimisation converges reliably and generally yields high-quality trajectories, but is rather expensive and can get stuck in local minima. We are exploring refinements to our approach, and how it may be combined with temporal explanation to provide a textual summary of simulated trajectories.

Experiments in a Higher-dimensional MDP

We now deploy TRIPLETREE in a more complex MDP: LUNARLANDERCONTINUOUS-V2 within OpenAI Gym (Brockman et al. 2016). Here, the state s is an 8-dimensional vector $[x, y, v_x, v_y, \phi, v_\phi, c_L, c_R]$, which are respectively the horizontal and vertical position and velocity, orientation, and angular velocity of a 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 vertical engine (-1 is off) and the second is a left-right side engine (0 is off). The reward is $+100$ for a safe landing in a landing zone and -100 for a crash, and there is additional shaping reward to disincentivise fuel burn. The black box target policy for our model is a Soft Actor-Critic deep RL agent from *Baselines Zoo* (Raffin 2018), the highest-performing policy on that repository.

Using a dataset of 10^5 observations, we grow a TRIPLETREE of up to 1000 leaves with $\theta = [1, 1, 1]$ (the multivariate action space requires us to slightly modify the action impurity measure). Figure 7 shows how the three losses vary during growth on both the training set and a validation set. In this more complex MDP the prediction problem is harder – particularly, it seems, for derivatives – and losses do not re-

duce to near zero, but as we shall see, the model still captures enough of the statistical properties of the system to deliver significant insight. We use the validation losses to inform early stopping and select the 450-leaf tree for evaluation.

With an 8D state space, it is nontrivial to create visualisations like those in figure 4. We suggest two ways forward: *projection* and *slicing*, which are detailed in Appendix C. In the former, we project leaf hyperrectangles onto a plane defined by two feature axes. Where multiple projections overlap, we compute a marginal value for the colouring attribute as a weighted average. This creates a *partial dependence plot* (PDP) of the attribute over the two features. Figure 8 contains a diagram of the method, and results from the 450-leaf tree. The upper five plots are PDPs for the x-y plane (landing zone shown in red). Notice how the main engine fires less at high altitudes. Sample density is high in a column above the landing zone, and on the ground where the policy makes slow positional corrections. The value and derivatives plots reveal that despite the MDP being symmetric, the agent obtains higher value when landing from the left, and takes a less curved route when doing so. The side engine plot has weaker trends, but the dark band around $y = 1$ (indicating the engine tends to fire to the left) may explain the wider landing approaches on the right. PDPs for the y-v_y plane show hard thresholds in main engine activation at $v_y = 0$ and $y \approx 1$, and a U-shaped vertical speed profile. In the ϕ-v_ϕ plane, we see that the side engine fires in an intuitive way to maintain stability, that value is highest when $v_\phi \approx 0$, and that the lander has pendulum-like dynamics aside from several leaves (purple) where v_ϕ jumps. These likely reflect rapid changes in side engine activation.

Slice visualisation involves taking an axis-aligned planar cross-section of $\mathcal{S}$, and displaying all intersected leaves as rectangles. This creates an individual conditional expectation (ICE) plot of the colouring attribute rather than a PDP, which is useful for illustrating counterfactual explanations for which the true state and minimal foil differ in ≤ 2 features. Examples are shown in figure 9. These plots not only display the minimal state change required to realise the foil condition, but reveal some of the surrounding state space, giving an indication of the counterfactual's robustness.

We can also simulate trajectories in this MDP and visualise them in 2D. Figure 10 contains some examples. Rather than showing a single trajectory between two leaves, we display *all* possible trajectories between leaves within a start zone (blue/orange) and end zone (red), demonstrating the distribution of paths taken by the agent. The first plot clarifies our prior observation that approaches from the right are wider, and shows that they occasionally miss the landing zone altogether. Thereafter, the lander must 'shuffle' along the ground into position; a major source of lost value. Similarly, the second plot confirms that the lander's vertical speed profile is U-shaped, and in fact very close to quadratic. The final plot is the most interesting. If rotated to the left ($\phi < 0.5$), the lander's return to a stable, neutral orientation is direct and overdamped. From the right, trajectories back to neutrality tend to overshoot; a classic indicator of a poorly-tuned controller. This is further evidence that despite obtaining high reward, the policy is chronically asymmetric.

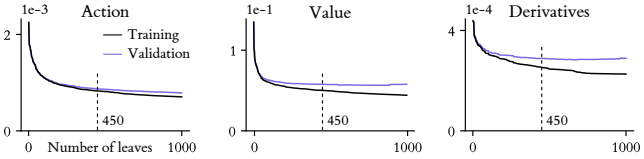


Figure 7: Training and validation losses in LUNARLANDER.

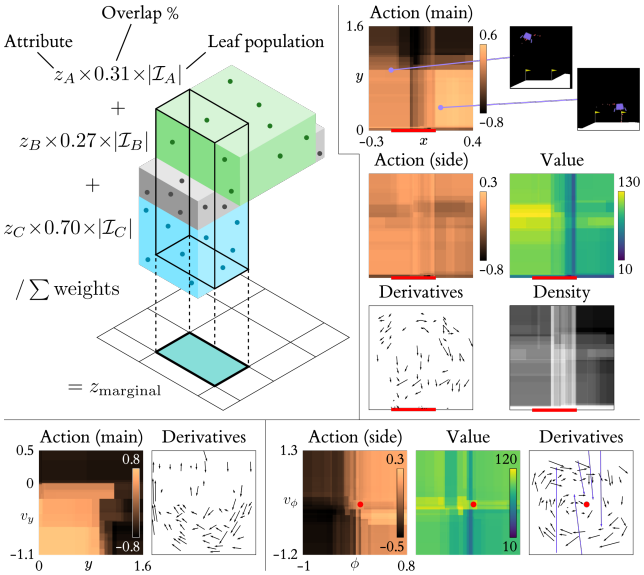


Figure 8: PDP projection method and results.

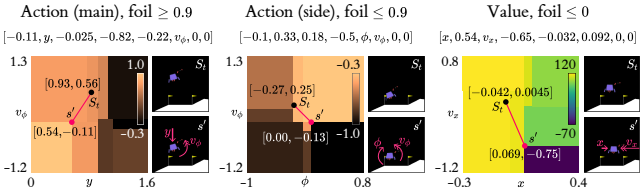
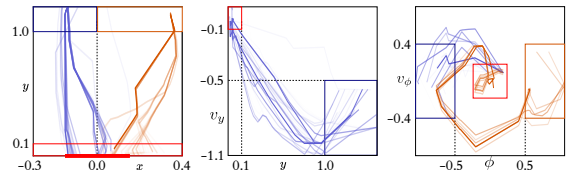


Figure 9: Displaying visual counterfactuals on ICE plots.


Figure 10: Simulated trajectories between regions of $\mathcal{S}$. The opacity of each trajectory is proportional to its probability.

Conclusion

Gärdenfors asserts that the aggregation of high-dimensional observational data into discrete convex regions, based similarity judgements, is a general route towards human understanding of complex systems. We consider TRIPLETREE to be a practical demonstration of this phenomenon; a versatile representational tool for delivering practical insight into the behaviour of black box autonomous agents through multivariate prediction, visualisation and rule-based explanation. In ongoing work, we continue to explore the potential of this representation, refining and expanding our methods of analysis and deploying the model in more challenging MDPs.

References

Adadi, A.; and Berrada, M. 2018. Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI). *IEEE Access* 6: 52138–52160.

Bastani, O.; Pu, Y.; and Solar-Lezama, A. 2018. Verifiable reinforcement learning via policy extraction. In *Advances in neural information processing systems*, 2494–2504.

Bewley, T.; Lawry, J.; and Richards, A. 2020. Modelling Agent Policies with Interpretable Imitation Learning. In *1st TAILOR workshop at ECAI 2020*.

Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. 1984. Classification and regression trees. Wadsworth & Brooks. *Cole Statistics/Probability Series* .

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym.

Carnap, R. 1967. *The logical structure of the world*. Routledge London.

Chomsky, N. 1959. A review of BF Skinner's Verbal Behavior. *Language* 35(1): 26–58.

Coppens, Y.; Efthymiadis, K.; Lenaerts, T.; Nowé, A.; Miller, T.; Weber, R.; and Magazzeni, D. 2019. Distilling deep reinforcement learning policies in soft decision trees. In *Proceedings of the IJCAI 2019 Workshop on Explainable AI*, 1–6.

De'Ath, G. 2002. Multivariate regression trees: a new technique for modeling species–environment relationships. *Ecology* 83(4): 1105–1117.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1(1): 269–271.

Edelman, S. 1998. Representation is representation of similarities. *The Behavioral and brain sciences* 21(4): 449.

Gärdenfors, P. 2004. *Conceptual spaces: The geometry of thought*. MIT press.

Guidotti, R.; Monreale, A.; Giannotti, F.; Pedreschi, D.; Ruggieri, S.; and Turini, F. 2019. Factual and Counterfactual Explanations for Black Box Decision Making. *IEEE Intelligent Systems* .

Jiang, W.-C.; Hwang, K.-S.; and Lin, J.-L. 2019. An Experience Replay Method based on Tree Structure for Reinforcement Learning. *IEEE Transactions on Emerging Topics in Computing* .

Kim, T.; Yue, Y.; Taylor, S.; and Matthews, I. 2015. A decision tree framework for spatiotemporal sequence prediction. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 577–586.

Koul, A.; Fern, A.; and Greydanus, S. 2018. Learning Finite State Representations of Recurrent Policy Networks. In *International Conference on Learning Representations*.

Lipton, P. 1990. Contrastive explanation. *Royal Institute of Philosophy Supplements* 27: 247–266.

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

Poyiadzi, R.; Sokol, K.; Santos-Rodriguez, R.; De Bie, T.; and Flach, P. 2020. FACE: feasible and actionable counterfactual explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, 344–350.

Pyeatt, L. D. 2003. Reinforcement learning with decision trees. In *21st IASTED International Multi-Conference on Applied Informatics*, 26–31.

Raffin, A. 2018. RL Baselines Zoo. https://github.com/araffin/rl-baselines-zoo. Version 1.6, released 22 Mar 2020, accessed 1 Aug 2020.

Rosch, E.; Mervis, C. B.; Gray, W. D.; Johnson, D. M.; and Boyes-Braem, P. 1976. Basic objects in natural categories. *Cognitive psychology* 8(3): 382–439.

Roth, A. M.; Topin, N.; Jamshidi, P.; and Veloso, M. 2019. Conservative Q-Improvement: Reinforcement Learning for an Interpretable Decision-Tree Policy. *arXiv:1907.01180* .

Saghezchi, H. B.; and Asadpour, M. 2010. Multivariate decision tree function approximation for reinforcement learning. In *International Conference on Neural Information Processing*, 687–694. Springer.

Samek, W.; Wiegand, T.; and Müller, K.-R. 2017. Explainable AI: Understanding, visualizing and interpreting deep learning models. *arXiv:1708.08296* .

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Thomas, P. S.; and Okal, B. 2015. A notation for Markov decision processes. *arXiv:1512.09075* .

Uther, W. T.; and Veloso, M. M. 1998. Tree based discretization for continuous state space reinforcement learning. In *AAAI/IAAI*, 769–774.

Wachter, S.; Mittelstadt, B.; and Russell, C. 2017. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.* 31: 841.

Wan, A.; Dunlap, L.; Ho, D.; Yin, J.; Lee, S.; Jin, H.; Petryk, S.; Bargal, S. A.; and Gonzalez, J. E. 2020. NBDT: Neural-Backed Decision Trees. *arXiv:2004.00221* .