What Did You Think Would Happen? Explaining Agent Behaviour through Intended Outcomes

Herman Yau [1] Chris Russell [2] Simon Hadfield [1]

Abstract

We present a novel form of explanation for Reinforcement Learning (RL), based around the notion of intended outcome. This describes what outcome an agent is trying to achieve by its actions. Given this definition, we provide a simple proof that general methods for post-hoc explanations of this nature are impossible in traditional reinforcement learning. Rather, the information needed for the explanations must be collected in conjunction with training the agent. We provide approaches designed to do this for several variants of Q-function approximation and prove consistency between the explanations and the Q-values learned. We demonstrate our method on multiple reinforcement learning problems.

1. Introduction

Explaining the behaviour of machine learning algorithms or AI remains a key challenge in machine learning. With the guidelines of the European Union's General Data Protection Regulation (GDPR) (EUd, 2018) calling for explainable AI, it has come to the research community's attention that it is vital to establish a clear understanding of black-box models. Despite significant research on explaining the behaviour of supervised machine-learning algorithms, it remains unclear exactly what should constitute an explanation for reinforcement learning. Current work in explainable reinforcement learning approaches it much like explaining a supervised classifier (Mott et al., 2019), and the explanations can highlight what in the current environment drives an agent to take an action, but not what the agent expects the action to achieve. Frequently, the consequences of the agent's actions are not immediate and a chain of many decisions all contribute to a single outcome. This paper addresses this problem by asking what chain of events the agent intended to happen as a result of a particular action choice. The importance of such explanations based around intended outcome in day to day life is well-known in psychology with Malle (2001) estimating that around 80% of these day to day explanations are intent-based. While the notion of intent makes little sense in the context of supervised classification, it is directly applicable to agent based reasoning, and it is perhaps surprising that we are the first work in explainable RL to directly address this.

Despite recent progress in reinforcement learning, few works have focused on its interpretability or explainability. Annasamy & Sycara (2019) proposed i-DQN, an architecturally similar model to the vanilla DQN model (Mnih et al., 2015) that introduced a key-value store to concurrently learn latent state representation, and Q-value mapping for intuitive visualisations by means of key clustering, saliency and attention maps. Mott et al. (2019) takes this idea further, representing the key-value store as recurrent attention model to produce a normalised soft-attention map, this made it possible to track the steps an agent takes to solve a task. Both methods focused on identifying components of the world that drive the agent to take a particular action, rather than identifying the agents expected outcomes. As these works exploit visual attention they are only directly applicable to image-based RL problems.

Methods have been proposed to extract global policy summaries, i.e. compressed approximations of the underlying policy function. Verma et al. (2018) proposed PIRL, which syntactically expresses policy with a high-level programming language. Topin & Veloso (2019) modelled policy as a Markov chain over abstract states giving rise to high-level explanations. Our approach gives a local description that says given a current state/action choice, what are the expected future states.

We present a novel yet simple addition to the standard RL framework which allows us to obtain a projection of possible future trajectories given a current observation and proposed action allowing for post-hoc interpretation of the agent's intention from based on its behavioural policy.

[1]Centre for Vision, Speech and Signal Processing, University of Surrey, Guildford, United Kingdom [2]Amazon, Tübingen, Baden-Württemberg, Germany. Part of this work was done while at the Alan Turing Institute and University of Surrey. Correspondence to: Herman Yau <h.yau@surrey.ac.uk>.

Workshop on Extending Explainable AI Beyond Deep Models and Classifiers, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

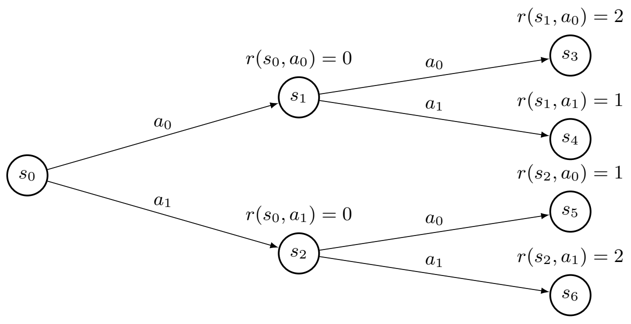


Figure 1: A simple MDP to illustrate state ambiguity in value iteration methods (see Section 2.1).

2. Value Function Decomposition

To understand why an RL agent prefers one action over another, we want to infer the agent's implicit estimate of $p(s_{t+n}|s_t, a_t, \pi)$ for all $n > 0$ which reveals the future states used in estimating Q-values.

We do not try to understand the internal computations that led our agent to map a particular Q-value against a particular action. Instead we wish to know the expected future states that leads an agent to decide that one action in the current state would be preferable to another action.

We prove that post-hoc explanations of the form we are interested in can not be recovered from an existing agent trained using a traditional Q-learning approach; we do this by showing that the inverse problem of recovering expected future states from a Q-function is fundamentally ill-posed. Note that Q-learning is used only as an example. The same reasoning can be applied to any value based reinforcement learning algorithm. We then propose a new algorithm that can be run alongside with an existing learning process which lets us obtain such explanations, without affecting what is learnt .

2.1. Value Ambiguity

The Q-value function is a convenient tool for producing a loss to guide the training of an agent. However, it acts as a bottleneck on the information exposed to both users and developers. It quantifies the "goodness" (expectation of discounted rewards) for executing a given action a in state s, without explicitly capturing *how* or *why* such a conclusion is reached. Furthermore, not only is the required information not directly available, but we also cannot in general, infer it indirectly from the Q-value.

Theorem 1. *Given any MDP where multiple optimal policies π^* exist, it is not possible to produce a post-hoc interpretation due to value ambiguity.*

The proof follows immediately by contradiction. Consider the undiscounted, deterministic chain of MDP in Figure 1, where an episode starts at s_0. Each state has 2 actions: a_0 moves to the upper node, a_1 moves to the lower node. All rewards are 0 except for reaching the end of the chain, where $r(s_1, a_0) = r(s_2, a_1) = 2$, $r(s_1, a_1) = r(s_2, a_0) = 1$. We assume a tabular Q-learning agent trained using the ϵ-greedy algorithm (with ties broken arbitrarily).

While the value functions are updated and eventually converge to optimality, there is ambiguity while learning takes place. Clearly, $Q^*(s_0, a_0) = Q^*(s_0, a_1) = 2$ and there are two optimal trajectories $\tau^1_{t:T} = (s_0, a_0, 0, s_1, a_0, 2)$ or $\tau^2_{t:T} = (s_1, a_1, 0, s_2, a_1, 2)$ relating to two separate policies π^*.

Clearly, in this situation it is not possible to examine a single value estimate $V^\pi(s_0)$ and infer which trajectory the agent expects to follow. Therefore we are unable to generate a sufficient post-hoc interpretation to reason about the agent's intention in this situation. □

Although this is a simple example, it immediately rules out approaches to generating explanations by training an ML method to predict to future behaviour of an agent on the basis of the behaviour of previous agents with similar Q-values. The problem illustrated by our counterexample is exacerbated if we work with an extended chain of conceptually similar MDP, as s_3 and s_6 could both contribute to the estimation of the same Q-value. Similarly, the same situation is observed for $Q^*(s_1, a_0)$ and $Q^*(s_2, a_1)$. More specifically, it is impossible to answer the contrastive question of "on what basis the agent learns about the quality of this state and/or state-action pair".

2.2. Explaining Tabular Reinforcement Learning

We describe how to augment standard RL learning methods with a simple additional process that allows us to learn a map of discounted expected states $\mathbf{H}$ (which we refer to as the *belief map*) concurrently with learning the Q-value function.

The intuition behind our approach is straightforward. While Q-value based methods estimate the expected future reward summed over all discounted expected states, our approach preserves additional information, and captures the discounted expected state at the same time. By choosing update rules for $\mathbf{H}$ to match the update equations of (1) we guarantee consistency between the Q-values and the expected future states. Subsection 2.3 proves that these expected states are consistent with the learnt Q-values. Below we present three variations of value function decomposition. For simplicity, we assume the methods are off-policy, but our approach is directly applicable to on-policy methods.

We define $\mathbf{H}(s, a) \in \mathbb{R}^{S \times A}$ as the expected discounted sum of future state visitations in a deterministic MDP which starts by executing action a from state s. As a notational convenience we use $1_{s,a} \in \mathbb{R}^{S\times A}$ to denote a binary indicator function 1 at position s, a and 0 elsewhere.

Q-Learning Every time the Q-value estimator is updated during learning, we update the corresponding entries of $\mathbf{H}$ to maintain consistency. This is a direct adaption of the standard Bellman equation to $\mathbf{H}$, that gives the update rule:

$$\begin{aligned}\mathbf{H}(s,a) \leftarrow \mathbf{H}(s,a) + \alpha\big(1_{s,a} \\ + \gamma\mathbf{H}(s_{t+1}, \arg\max_{a\in\mathcal{A}} Q(s_{t+1},a)) - \mathbf{H}(s,a)\big)\end{aligned} \quad (1)$$

We update all state-action pairs in trajectory τ.

Monte Carlo Control Adapting the update to Monte Carlo control methods is straightforward. We simply replace $1_{s,a}$ with a vector sum of discounted state visitations:

$$\mathbf{H}(s,a) \leftarrow \mathbf{H}(s,a) + \alpha\left(\sum_{t=t'}^{T} \gamma^t 1_{s_t,a_t} - \mathbf{H}(s,a)\right) \quad (2)$$

2.3. The Consistency of Belief Maps

We say that a belief map is consistent with a Q-table given reward map $\mathbf{R}$, if the inner product of the belief map of every state-action pair with the reward map gives the Q-value for the same state-action pair. More formally, we assume the current reward is a deterministic function of the current state-action pair, we define $\mathbf{R} \in \mathbb{R}^{S\times A}$ as a map of rewards for every state-action pair in the MDP, and say that if

$$\mathrm{vec}(\mathbf{H}(s,a))^\top \mathrm{vec}(\mathbf{R}) = Q(s,a) \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \quad (3)$$

then the belief map $\mathbf{H}$ is consistent with Q.

Theorem 2. *Assuming that* $\mathbf{H}$ *and* Q *are zero-initialized, then* $\mathbf{H}$ *and* Q *will be consistent for all iterations of the algorithm.*

Proof. Proof follows by induction on i the iteration of the learning algorithm. **Base case** ($i = 0$): At initialization $\mathbf{H}(s,a) = \mathbf{0}$ and $Q(s,a) = 0$ for all possible s and a and the statement is trivially true.

Inductive step (Monte Carlo): We consider iteration i where we know that the theorem holds for time $i-1$. We use $\mathbf{H}_i$ to indicate the state of $\mathbf{H}$ at iteration i, and Q_i the state of Q at iteration i

$$\begin{aligned}&\mathrm{vec}(\mathbf{H}_i(s,a))^\top \mathrm{vec}(\mathbf{R}) \\ &= \mathrm{vec}((1-\alpha)\mathbf{H}_{i-1}(s,a) + \alpha\sum_{t=t'} r^t\gamma^t 1_{s_t,a_t})^\top \mathrm{vec}(\mathbf{R}) \\ &= (1-\alpha)Q_{i-1}(s,a) + \alpha\mathrm{vec}\left(\sum_{t=t'}^{T}\gamma^t 1_{s_t,a_t}\right)^\top \mathrm{vec}(\mathbf{R}) \\ &= (1-\alpha)Q_{i-1}(s,a) + \alpha\sum_{t=t'}^{T}\gamma^t r_t \\ &= Q_i(s,a)\end{aligned} \quad (4)$$

Inductive step (Q-learning):We consider time i where we know that the theorem holds for time $i-1$. We write $m_t = \arg\max_{a\in\mathcal{A}} Q_{i-1}(s_t, a)$

$$\begin{aligned}&\mathrm{vec}(\mathbf{H}_i(s,a))^\top \mathrm{vec}(\mathbf{R}) \\ &= \mathrm{vec}((1-\alpha)\mathbf{H}_{i-1}(s,a) \\ &+ \alpha\left(1_{s,a} + \gamma\mathbf{H}_{i-1}(s_{t+1}, m_{t+1})\right))^\top \mathrm{vec}(\mathbf{R}) \\ &= (1-\alpha)Q_{i-1}(s,a) \\ &+ \alpha\mathrm{vec}\left(1_{s,a} + \gamma\mathbf{H}_{i-1}(s_{t+1}, m_{t+1})\right)^\top \mathrm{vec}(\mathbf{R}) \\ &= (1-\alpha)Q_{i-1}(s,a) + \alpha Q_{i-1}(s_{t+1}, m_{t+1}) \\ &= Q_i(s,a).\end{aligned} \quad (5)$$

as required. □

2.4. Deep Q-Learning

Much like Deep Q-learning, function approximation can be used for estimating $\mathbf{H}$. For $\mathbf{H}$ parametrised by θ_h and given agent policy π_θ, the loss function is:

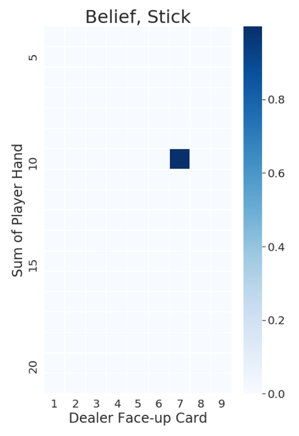
$$\begin{aligned}L_h(\theta_{h_i}) = &\mathbb{E}_{(s,a,1,s_{t+1})}[(1_{s,a} \\ &+ \gamma\mathbf{H}(s, \arg\max_{a\in\mathcal{A}} Q(s_{t+1}, a; \theta_i); \theta^-_{h_i}) \\ &- \mathbf{H}(s,a;\theta_{h_i}))]\end{aligned} \quad (6)$$

then the gradient update is:

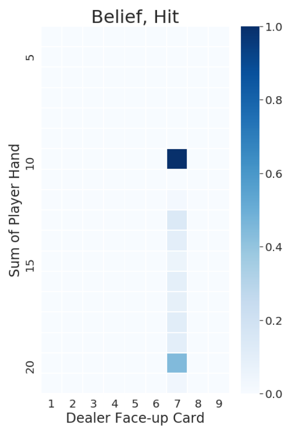
$$\theta_{h_{i+1}} = \theta_{h_i} + \alpha\nabla_{\theta_h} L(\theta_{h_i}). \quad (7)$$

Here $\mathbf{H}(s,a)$ is an unnormalised density estimation. Where the state and action space is discrete this can be in the form of a vector output, and where at least one is continuous using techniques such as (De Cao et al., 2019) or approximately by pre-quantizing the state space using standard clustering techniques.

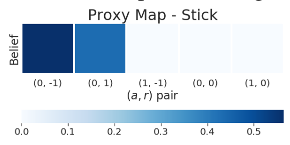
The mathematical guarantees of consistency from the previous section do not hold for deep Q-learning, with the random initialisation of the networks meaning that it fails in the base case. However, just as an appropriately set up deep



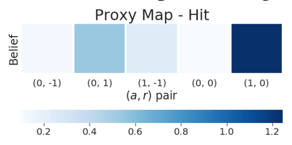
(a) Belief map for sticking



(b) Belief map for hitting

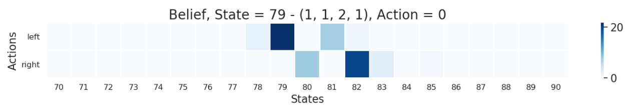


(c) Reward belief for sticking

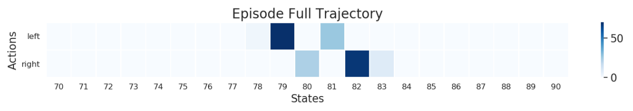


(d) Reward belief for hitting

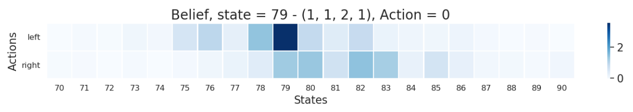
Figure 2: Visualisations of belief maps and reward incurred by actions when player sum = 10, dealer card = 7 with no usable ace. Unreachable player states are ignored for conciseness, i.e. Player sum < 4 and sum > 21.



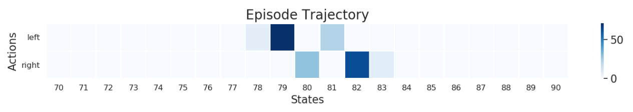
(a) Belief map for Q-Learning



(b) True trajectory for Q-learning



(c) Belief map for deep Q-Learning



(d) True trajectory for deep Q-learning

Figure 3: Belief maps and trajectories visualisations for cartpole simulation.

Q-learning algorithm will converge to an empiric minimiser, an appropriately setup H-learner will also converge to minimise Equation (6). We evaluate the deep variants of our approach in the experimental section, and show even without theoretical guarantees, it performs effectively, providing insight into what has been learnt by our agents.

3. Experiments and Discussions

We evaluate our approach on three standard environments using OpenAI Gym (Brockman et al., 2016) - Blackjack, Cartpole (Barto et al., 1990) and Taxi (Dietterich, 2000). Each environments poses unique challenges. For each task we briefly describe our assumptions and key experiment parameters. We extract an intuitive post-hoc interpretation of the RL agent's intention solely from the belief maps. We verify the correctness of our implementation in each environment using theorem 2, and confirm that Equation (3) holds.

We find minimal differences applying our technique to Temporal Difference and Monte Carlo approaches. For conciseness, we report here the findings from TD and the MC results will be provided as a supplementary report. RL agents always use an ϵ-greedy algorithm during training.

Blackjack We model a simplified game of blackjack and assume (1) the shoe consists of infinite cards, (2) the only moves allowed are "hit" and "stick" and (3) we do not account for blackjacks.

The system dynamics are non-deterministic as the card drawn on each hit is random, and environment the reward is stochastic due to the uncertainty in the dealer's hand. To make the reward a deterministic function of the stochastic state we create five additional states - hit and bust, stuck and won, stuck and drew, stuck and lost and hit only. The agent is trained with $\alpha = 0.1, \gamma = 1$ for $500k$ episodes.

Figure 7 shows belief maps for each action and the corresponding incurred rewards when the player's hand has a sum of 10, and the dealer shows 7. There is a single point in the state space that shows a high probability in both figure 2a and 2b. This point corresponds to the current starting state, which all future trajectories must pass through. Figure 2b shows probable future events when the player hits. We observe the intention of the agent from the belief map, that it continues to hit until a high enough sum is obtained. This explains why lower values have lower density, as there are fewer paths to reach these states. The second peak at 20 is expected given that all face cards have a value of 10.

Cartpole (Barto et al., 1990) is a classic continuous control problem with a 4-tuple continuous state space $\langle x, \dot{x}, \theta, \dot{\theta} \rangle$: cart position, cart velocity, pole angle and pole velocity, and a discrete action space consisting of 2 actions: left and right. For a like-with-like comparison across standard learning approaches, we discretizing the state-space into bins of $(3, 3, 6, 3)$.

We investigate agent intention in cartpole learnt from vanilla Q-learning and DQN. In both methods, we train the agent until the agent is reasonably stable. The vanilla Q-learning agent is trained with learning rate $\alpha = 0.1$ and $\gamma = 1$. The DQN agent is trained using Adam (Kingma & Ba, 2014) with gradient clipping at $[-1, 1]$ and $\alpha = 0.0001, \gamma = 1$, while the belief map is trained by a deep network which we refer to as a *Deep Belief Network* (DBN) using the same hyperparameters.

Figure 3 shows the belief map versus the actual trajectory of the episode produced by the agent. The belief maps show that the agent identified two key states to reach in order to achieve stability. The high belief in both states indicates the agent's intention to oscillate between these two states. This is reflected by the historic trajectory map where the same pattern is seen indicating that the agent is successfully execute this plan. For DQN, the belief map is noticeably less sharp. Nevertheless it closely resembles the trajectory that is achieved by the DQN agent, and reveals the agent's intention.

Taxi (Dietterich, 2000) is a challenging environment where an agent must first navigate from a randomized start point, around obstacles, to collect a passenger from a random location, then pick up that passenger and navigate to a randomized destination. We investigate agent intention learnt by Q-learning and DQN. The Q-learning agent is trained with $\alpha = 0.4, \gamma = 1$ using a 4-tuple state representation of car x and y location, passenger location and destination. DQN used $\alpha = 0.0001, \gamma = 1$ for $100k$ episodes.

Figure 4 and Figure 5 shows visualisations of the RL agents' belief maps at several steps along a trajectory. For both methods we can intuitively reason about the intentions of the agents. These intentions become clearer over time, as the multiple potential routes (including the outward and return journey) which are overlaid, collapse to the single shortest path towards the goal.

It is interesting to note that the intention of the DQN agent appears much less diffuse than those of the Q-learning agent over long periods of time. This reflects the increased effectiveness of the DQN agent in solving the task. However, we can also note that the DQN agent appears to be exhibiting a bias towards returning to the top left corner position even though the goal is in the bottom left. Having access to this type of insight would be useful in debugging and identifying problems in learning.

Contrastive explanations Figure 6 demonstrates the capability of our method to extract contrastive explanations. To generate these explanations, we scaled the belief map to $[0, 1]$ and weight images of non-zero states accordingly. Figure 6a reveals the value ambiguity problem stated previously, in a real example. By contrasting the intentions of the best and second best actions we can observe two equally good policies were produced. Such explanation is not possible with vanilla Q-learning which only has access to Q-values. Similarly in Figure 6b, despite the suboptimality of the second best action, we can identify that the agent is able to recover by bouncing back to the original position immediately then following the same route to the destination. In both cases we can reason about the agent's intention: that it is indeed solving the assigned task. We also identify a failure case in Figure 6c where taking the suboptimal action would cause the car to be stuck in the same state. In DQN, even though the mathematical guarantee is lost, we can nevertheless extract an approximation of the agent's intention.

4. Conclusions and Future Work

We have proposed a novel approach for explaining what outcomes are implicitly expected by reinforcement learning agents. We proposed a meaningful definition of intention for RL agents and proved no post-hoc method could generate such explanations. We proposed modifications of standard learning methods that generate such explanations for existing RL approaches, and proved consistency of our approaches with tabular methods. We further showed how it can be extended to deep RL techniques and demonstrated its effectiveness on multiple reinforcement learning problems.

A noticeable limitation of our method is that it is best-suited for problems where tabular reinforcement learning works well (i.e. low-dimensional and easily visualisable). Just as deep learning substantially increased the applicability of reinforcement learning it is interesting to ask how it could increase the applicability of our approach. One potential answer lies in the use of concept activation vectors(Kim et al., 2017), which allows for shared concepts between humans and machine learning algorithms. For example, if a concept was trained to recognise a pinned piece in chess our approach would be able to explain a move by saying that it will give rise to a pinned knight later. Importantly, given a set of concepts F defined over the state space $\mathcal{S}$, it would be possible to train the belief map over the low-dimensional space $F(\mathcal{S})$, rather than $\mathcal{S}$, substantially improving the scalability of our approach. As such, while our approach is directly applicable to problems with easily mappable state-spaces, it also lays the mathematical foundations for explainable agents in more complex systems.

Acknowledgements

This work was partially supported by the UK Engineering and Physical Sciences Research Council (EPSRC), Omidyar Group and The Alan Turing Institute under grant agreements EP/S035761/1 and EP/N510129/1.

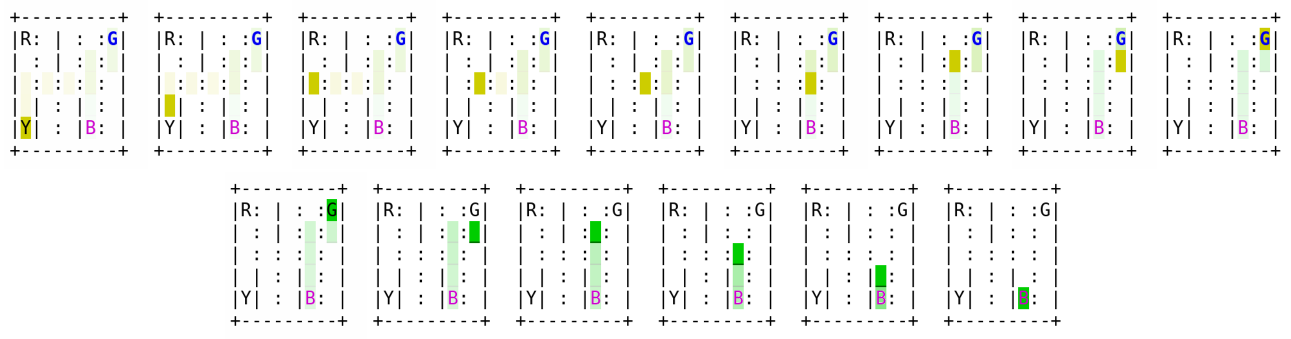


Figure 4: Varying intention of the Q-learning agent in the Taxi environment (Dietterich, 2000). Each image represents the belief map at one point along a trajectory, with time advancing from left to right. Colour intensity denotes the confidence that the agent will visit a state. Yellow indicates the passenger is not present, whereas green means they are. 'R', 'G', 'Y', 'B', each represents a possible location of the passenger and the destination. Blue text indicates the location of the passenger if they are not in the car, and red text denotes destination.

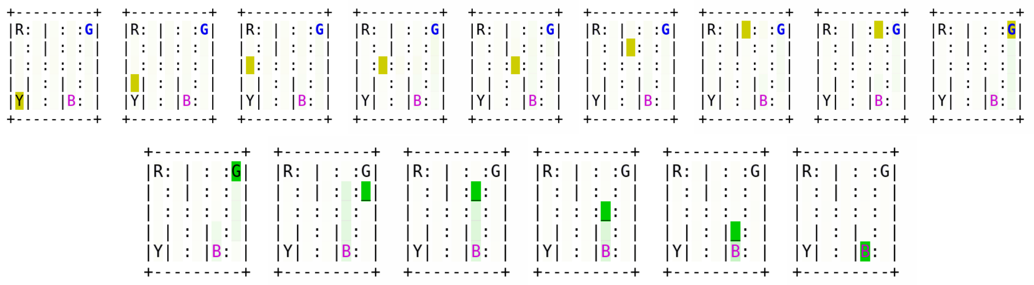
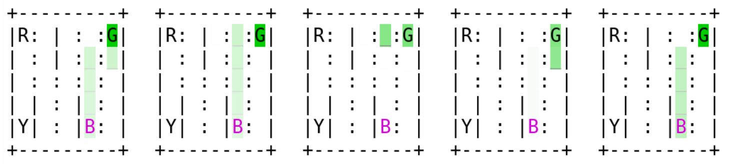
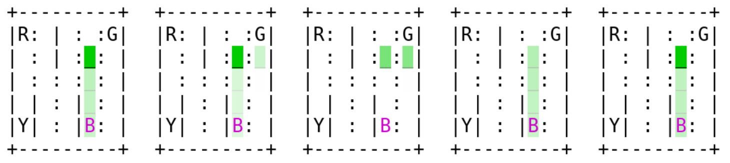


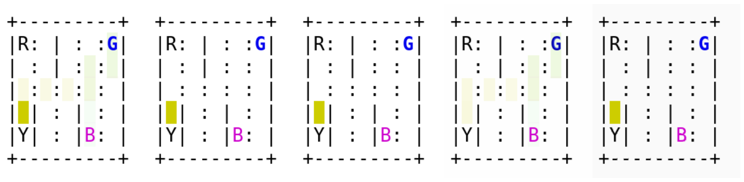
Figure 5: How the expected outcomes of the DQN agent vary in the Taxi environment (Dietterich, 2000)



(a) a_0: move east, a_1: move south. $Q(s, a_0) = 7.7147$, $Q(s, a_1) = 7.0777$.



(b) a_0: move south, a_1: move east. $Q(s, a_0) = 11.87$, $Q(s, a_1) = 7.6599$.



(c) a_0: move north, a_1: move east. $Q(s, a_0) = -2.3744$, $Q(s, a_1) = -5.204$.

Figure 6: Contrastive explanations of taxi tabular Q-learning agent during the episode in Figure 4. Here we denote a_0 as the best action, a_1 as the second best action. Current state is highlighted by the strong hue of the car. From left to right we have: (1) $\mathbf{H}(s, a_0)$, (2), $\mathbf{H}(s, a_1)$, (3) $\min(0, \mathbf{H}(s, a_1) - \mathbf{H}(s, a_0))$, (4) $\min(0, \mathbf{H}(s, a_0) - \mathbf{H}(s, a_1))$, (5) the intersection of the two expected outcomes, given by $\min(\mathbf{H}(s, a_0), \mathbf{H}(s, a_1))$.

References

2018 reform of eu data protection rules, 2018. URL https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf.

Annasamy, R. M. and Sycara, K. Towards Better Interpretability in Deep Q-Networks. Technical report, 2019. URL www.aaai.org.

Barto, A. G., Sutton, R. S., and Anderson, C. W. *Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems*, pp. 81–93. IEEE Press, 1990. ISBN 0818620153.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

De Cao, N., Titov, I., and Aziz, W. Block neural autoregressive flow. *arXiv preprint arXiv:1904.04676*, 2019.

Dietterich, T. G. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.*, 13(1):227–303, November 2000. ISSN 1076-9757.

Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., and Sayres, R. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). *arXiv preprint arXiv:1711.11279*, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2014.

Malle, B. F. Folk explanations of intentional action. *Foundations of social cognition*, 2001.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Mott, A., Zoran, D., Chrzanowski, M., Wierstra, D., and Rezende, D. J. Towards Interpretable Reinforcement Learning Using Attention Augmented Agents. Technical report, 2019.

Topin, N. and Veloso, M. Generation of Policy-Level Explanations for Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:2514–2521, 2019. ISSN 2159-5399. doi: 10.1609/aaai.v33i01.33012514. URL www.aaai.org.

Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. Programmatically interpretable reinforcement learning. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5052–5061. PMLR, 2018. URL http://proceedings.mlr.press/v80/verma18a.html.

A. Agent Description

Blackjack In both Monte Carlo control and Q-learning we share the same training settings. We set the learning rate $\alpha = 0.1$, discount factor $\gamma = 1$. We set an initial exploration probability $\epsilon = 1$ which is exponentially decreased to $\epsilon = 0.05$ with a decay rate of 0.9999 throughout the training.

Cartpole Similar to blackjack, we share the same training settings in cartpole. We set learning rate $\alpha = 0.1$ and discount factor $\gamma = 1$. Episode terminates when the length of the episode reaches 200 timesteps. We initially set exploration probability $\epsilon = 1$ which is linearly decreased to $\epsilon = 0.1$ throughout the first 500 episodes.

In DQN training, we use Adam(Kingma & Ba, 2014) optimizer with $\epsilon = 1e - 8$, exponential decays $\beta_1 = 0.9, \beta_2 = 0.999$. The learning rate is $\alpha = 0.0001$. We use Huber loss with discount factor $\gamma = 1$. We clip gradients to be in the range of $[-1, 1]$. For each learning iteration, we batch 16 experience together for optimisation. We initially set exploration probability $\epsilon = 1$ which is linearly decreased to $\epsilon = 0.1$ throughout the first 500 episodes. Full details of the neural network architectures can be found in Table 1 and 2.

Layer	Type	Input	Size
input	N/A	N/A	4
fc1	MLP	input	128
fc2	MLP	fc1	512
output	MLP	fc2	2

Table 1: DQN neural network architecture

Layer	Type	Input	Size
input	N/A	N/A	162
fc1	MLP	input	512
fc2	MLP	fc1	1024
fc3	MLP	fc2	2048
output	MLP	fc3	2*2*162

Table 2: DBN neural network architecture. Output is reshaped to $\mathbb{R}^{A \times S \times A}$ before return.

Taxi We first describe the training details for Q-learning. We set learning rate $\alpha = 0.4$ and discount factor $\gamma = 0.9$. Episode terminates agent completes the episode or reach a threshold of 200 timesteps. We initially set exploration probability $\epsilon = 1$ which is linearly decreased to $\epsilon = 0.1$ in the first 250 episodes.

In DQN training, we use Adam(Kingma & Ba, 2014) optimizer with $\epsilon = 1e - 8$, exponential decays $\beta_1 = 0.9, \beta_2 = 0.999$. The learning rate is $\alpha = 0.0001$. We use Huber loss with discount factor $\gamma = 1$. We clip gradients to be in the range of $[-1, 1]$. For each learning iteration, we batch 16 experience together for optimisation. We initially set exploration probability $\epsilon = 1$ which is linearly decreased to $\epsilon = 0.1$ in the first 250 episodes. In order to speed up training of DBN, we made a small modification by splitting belief update and action into two inputs: belief update is a binary indicator function 1 at position s, and action is converted into one-hot encoding before being fed into the neural network. Full details of the neural network architectures can be found in Table 3 and 4.

Layer	Type	Input	Size
input	N/A	N/A	4
fc1	MLP	input	500
fc2	MLP	fc1	2000
output	MLP	fc2	6

Table 3: DQN neural network architecture

Layer	Type	Input	Size
input_belief	N/A	N/A	500
input_action	N/A	N/A	6
belief_stream	MLP	input_belief	1024
action_stream	MLP	input_action	128
concat	N/A	input_belief, input_action	1024+128 = 1152
fc3	MLP	concat	2048
output	MLP	fc3	500

Table 4: DBN neural network architecture. Output is reshaped to $\mathbb{R}^{A \times S \times A}$ before return.

B. Further Results

Suppose we have the best action a_0 and second best action a_1 at state s, we can compute a contrastive explanation G by subtracting the belief of a_0 against a_1:

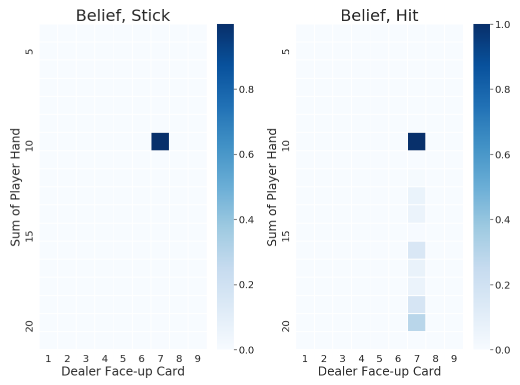
$$G(s, a_0, a_1) = \mathbf{H}(s, a_0) - \mathbf{H}(s, a_1) \tag{8}$$

Since $\mathbf{H}(s, a)$ is a decomposition of $Q(s, a)$, the subtraction will tell us which future states constitute the overall goodness of a_0 over a_1 in s. We apply equation 8 to generate contrastive explanations below.

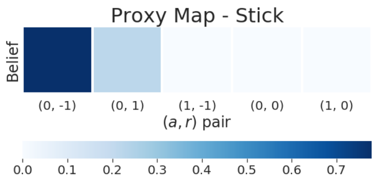
Blackjack For conciseness, the visualisations of blackjack have been trimmed to only show reachable states. We verify in Figure 9 and Figure 8 that we can recover the Q-table from learned beliefs via a proxy reward map, thus our theorem in the main paper holds. In figure 10 and 11 we show that different contrastive explanations can be extracted by applying equation 8.

Cartpole We verify in figure 13 and 14 that our theorem in the main paper holds without the aid of a proxy map. We also give a demonstration of contrastive explanations in figure 15 and 16.

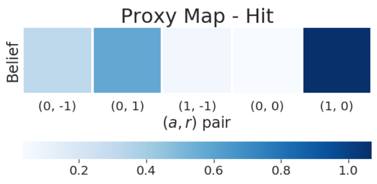
Taxi We provide animations of Figure 4 and 5 in the main text; see attached video for the animation. We also give contrastive explanations from DQN agent in Figure 17. Although the contrastive explanations are noticeably fuzzier, the extracted intention is similar to our results for Q-learning.



(a) Belief map for sticking (b) Belief map for hitting



(c) Reward belief for sticking



(d) Reward belief for hitting

Figure 7: Monte Carlo control blackjack simulation: here we show the visualisations of belief maps and proxy action-reward map when player sum $= 10$, dealer card $= 7$ with no usable ace.

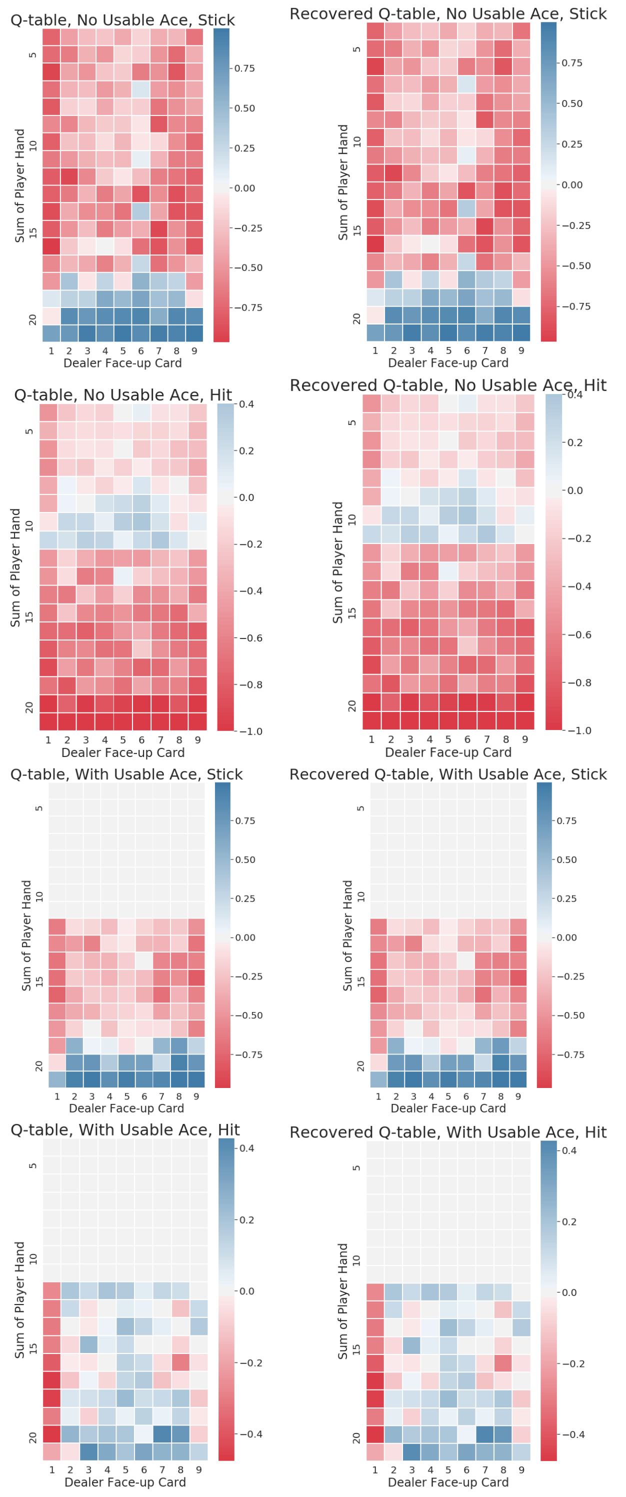


Figure 8: Q-learning: visualisations of ground truth Q-table and recovered Q-table from learned belief.

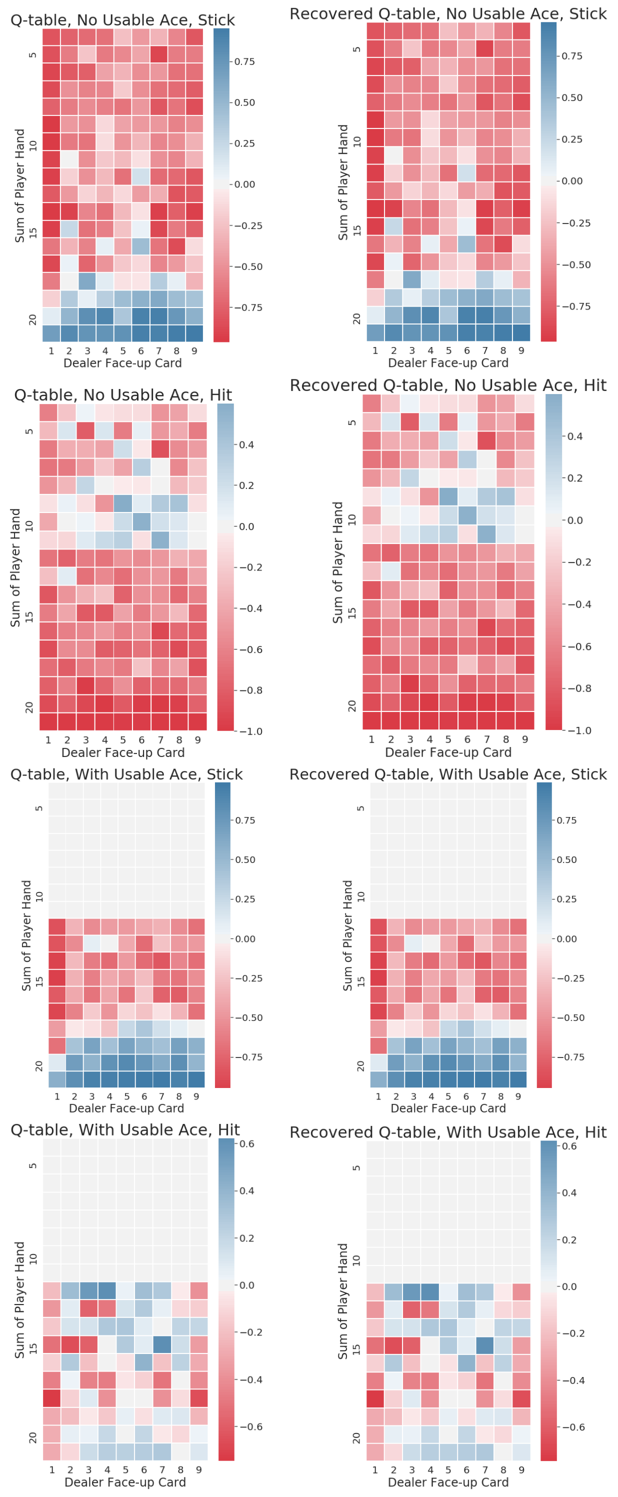
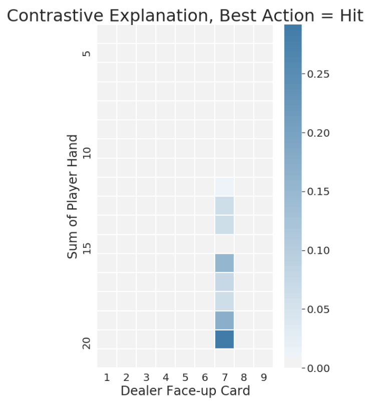
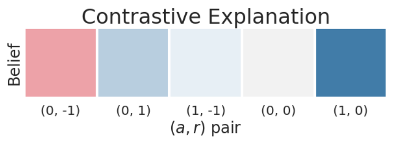


Figure 9: Monte Carlo control: visualisations of ground truth Q-table and recovered Q-table from learned belief.

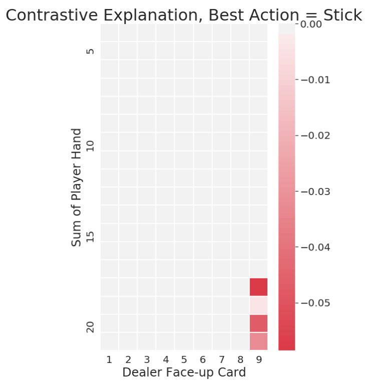


(a)

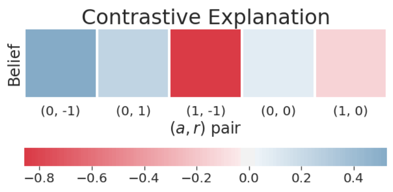


(b)

Figure 10: Monte Carlo control, blackjack simulation: contrastive explanation when player sum $= 10$, dealer card $= 7$ with no usable ace. Figure 10a shows that hitting would grant access to various future states. This is reflected in the figure 10b, as the red block (consequence of sticking) indicates sticking is undesirable since the action is more likely to generate -1 reward.

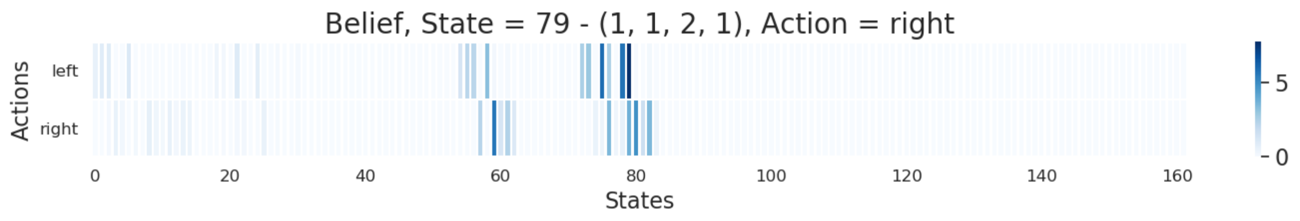


(a)

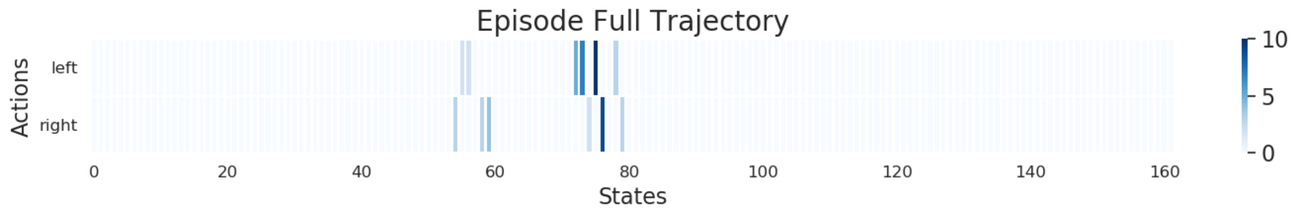


(b)

Figure 11: Monte Carlo control, blackjack simulation: contrastive explanation when player sum $= 17$, dealer card $= 9$ with no usable ace. In contrast to figure 10a, hitting is more likely to yield a more negative consequnce than twisting, as evidenced by 11a that the red block (consequence of hitting) at $(1, -1)$ clearly outweights the cumulative sum of blue blocks (consequence of sticking).



(a) Belief map



(b) True trajectory

Figure 12: Monte Carlo control, cartpole simulation: belief map and trajectory visualisations.

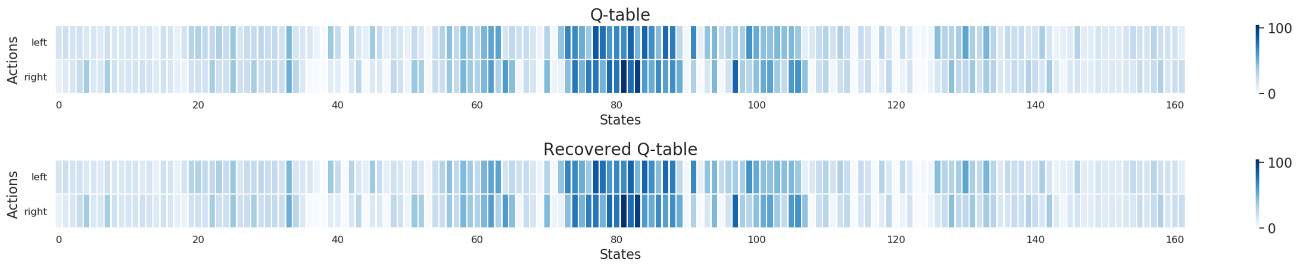


Figure 13: Monte Carlo control, cartpole simulation: Q-table and recovered Q-table from learned beliefs. Each value is numerically identical except for floating-point errors.

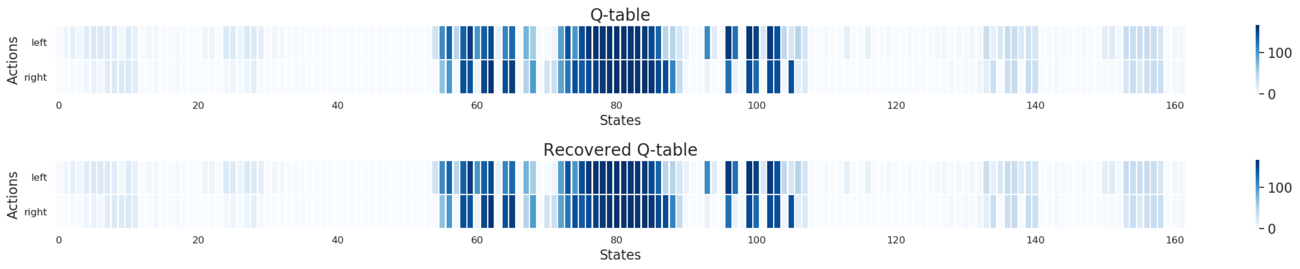


Figure 14: Q-learning, cartpole simulation: Q-table and recovered Q-table from learned beliefs. Each value is numerically identical except for floating-point errors.

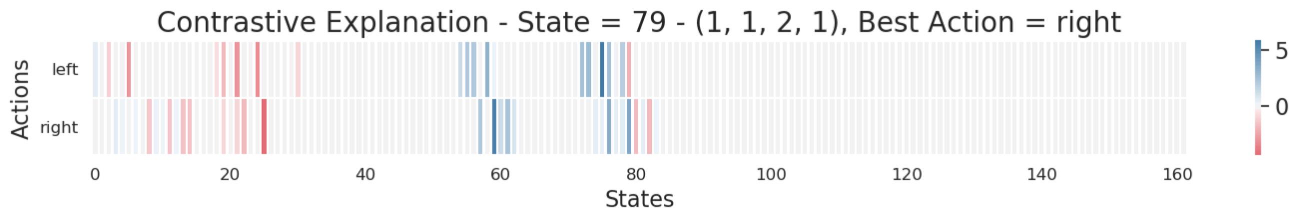


Figure 15: Monte Carlo control, cartpole simulation: selected contrastive explanations at state 79. We can intuitively reason and justify the agent's decision of moving to the right since it offers a greater stability. Moving to the left risks heading towards the edges highlighted by the red blocks.

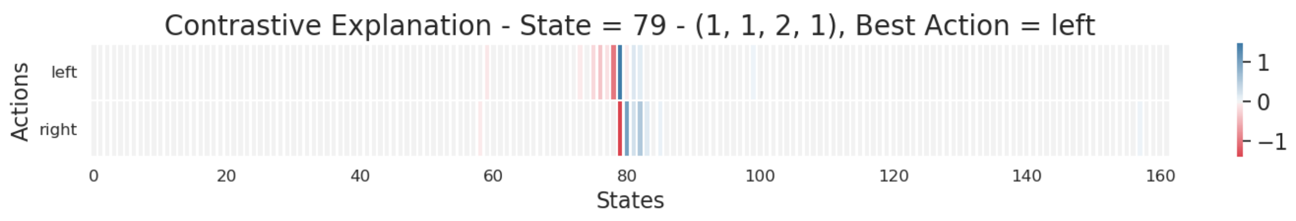
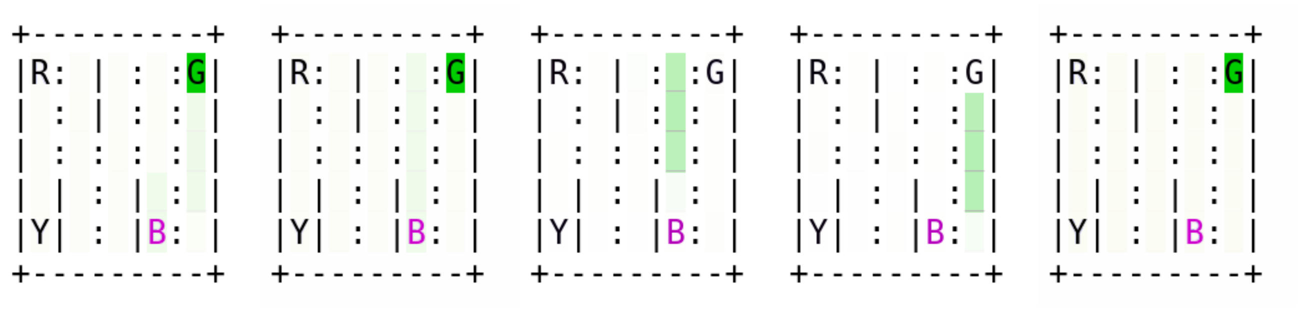
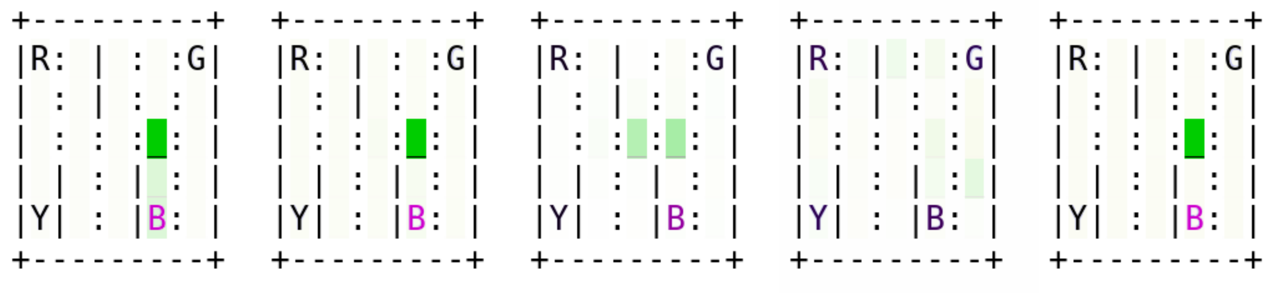


Figure 16: Q-learning, cartpole simulation: selected contrastive explanations at state 79. The figure shows that moving to the right is undesirable since it will lead the agent to visit states which can cause possible failure, whereas moving to the left can offer more guarantee of staying in the middle.



(a) Value ambiguity. a_0: move south, a_1: move west. $Q(s, a_0) = 14.9908$, $Q(s, a_1) = 14.9858$.



(b) Bounce back. a_0: move south, a_1: move west. $Q(s, a_0) = 17.9323$, $Q(s, a_1) = 15.9981$.

Figure 17: DQN, taxi simulation: contrastive explanations of taxi DQN agent. Refer to figure 6 in main text for descriptions of each subplot.