ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240814

Explaining an Agent's Future Beliefs Through Temporally Decomposing Future Reward Estimators

Mark Towers[a,*], **Yali Du**[b], **Christopher Freeman**[a] **and Timothy J. Norman**[a]

[a]School of Electronics and Computer Science, University of Southampton, UK
[b]Department of Informatics, Kings College London, UK
ORCID (Mark Towers): https://orcid.org/0000-0002-2609-2041, ORCID (Yali Du): https://orcid.org/0000-0001-5683-2621, ORCID (Christopher Freeman): https://orcid.org/0000-0003-0305-9246, ORCID (Timothy J. Norman): https://orcid.org/0000-0002-6387-4034

Abstract. Future reward estimation is a core component of reinforcement learning agents; i.e., Q-value and state-value functions, predicting an agent's sum of future rewards. Their scalar output, however, obfuscates when or what individual future rewards an agent may expect to receive. We address this by modifying an agent's future reward estimator to predict their next N expected rewards, referred to as Temporal Reward Decomposition (TRD). This unlocks novel explanations of agent behaviour. Through TRD we can: estimate when an agent may expect to receive a reward, the value of the reward and the agent's confidence in receiving it; measure an input feature's temporal importance to the agent's action decisions; and predict the influence of different actions on future rewards. Furthermore, we show that DQN agents trained on Atari environments can be efficiently retrained to incorporate TRD with minimal impact on performance.

1 Introduction

With reinforcement learning (RL) agents exceeding human performance in complex and challenging game environments (e.g., Atari 2600 [3], DotA 2 [6], and Go [22]), there is significant interest in applying these methods to address practical problems, often in support of human judgement. There are several barriers to realising this vision, however, with the need for agents to be able to explain their decisions one of the most important [18]; agents need to be able to work with people [7], and so we need effective Explainable Reinforcement Learning (XRL) mechanisms.

Central to RL agents is a future reward estimator (Q-value or state-value function) predicting the sum of future rewards for a given state. These functions are used either explicitly in the policy itself (e.g., DQN [16]) or for learning with a critic (e.g., PPO [19] and TD3 [8]). However, few XRL algorithms have devised methods to explain these functions directly. One problem is that their scalar outputs provide no information on its composition (i.e., when and what future rewards the agent believes it will receive), just its expected cumulative sum.

An example of this problem is illustrated in Figure 1 where a drone has two paths: up or down. Depending on the path taken, the drone can receive coins for 1 point each or the treasure chest for 4 points. Using a discount factor of 0.95, the drone's Q-value for moving up is 3.26 while moving down is 3.52. Despite this small difference in Q-values, the quantity and temporal immediacy of their expected rewards are radically different; moving up, the drone receives a single large reward, while moving down receives many smaller rewards. A fact unknown from observing the Q-values alone although critical to agent behaviour in selecting whether to move up or down.

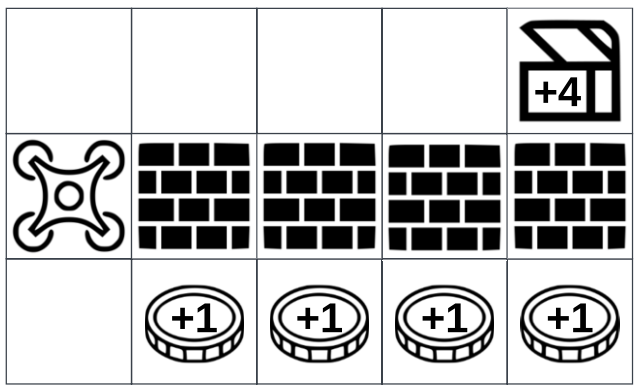

Figure 1. Example Gridworld with an agent and two paths (up and down) that contain different rewards.

We propose a novel future reward estimator that predicts the agent's next N expected rewards for a given state, referred to as Temporal Reward Decomposition (TRD) (Section 4). We prove that TRD is equivalent to the Q-value and state-value functions. In this way, TRD can report the temporal immediacy and quantity of future rewards for different action choices, enabling decisions to be explained and contrasted. For example, using Figure 1, the agent's TRD Q-value for moving down is $[0, 0.95, 0.90, 0.86, 0.81]$ and moving up $[0, 0, 0, 0, 3.26]$, enabling us to produce explanations such as "while the sum of actual rewards is equal, taking the route down has more immediate rewards, which are preferred by the drone due to its discount factor".

Implementing TRD requires only two changes to a deep RL agent's future reward estimator: increase the network output by $N+1$ for predicting the future rewards and a novel element-wise loss function of future rewards (Section 4). Importantly, TRD can achieve

* Corresponding Author. Email: mt5g17@soton.ac.uk

similar performance as DQN [16] agents for Atari environments [4] across a wide range of N (Section 5).

Building on this direct access to an agent's predictions for individual future rewards, we explore three novel applications for understanding an agent's decision-making (Section 6). The first is to generate explanations for an action choice based on when and what rewards it will receive, and, for particular environments, the agent's confidence in collecting a reward (Section 6.1). The second is to explain how the importance of an observation feature changes depending on how far into the future a reward is expected; e.g., identifying features that are more activate for earlier rewards (Section 6.2). Thirdly, we produce contrastive explanations using the difference in future expected rewards for two actions, which can reveal changes in expected rewards the agent will receive between them (Section 6.3). Together, these three explanation mechanisms demonstrate the value of Temporal Reward Decomposition for XRL.

Prior to presenting TRD, proving key properties and detailing how this can be used to generate explanations of an agent's action choices with respect to future rewards, we briefly review relevant prior research and present some preliminary formalisation that we build upon.

2 Related Work

In this brief review we focus on existing reward-based explanation mechanisms and algorithms for understanding an agent's decision-making in similar domains (see Qing *et al.* [18] for a survey).

Prior work in XRL has explored decomposing the Q-value into reward components and by future states. Juozapitis *et al.* [11] proposed decomposing the future rewards into components. In Figure 1, for example, we have two components (or reward sources), the treasure chest and the coins. An explanation would then contrast the coin(s) versus treasure chest(s) along different trajectories, but not when any rewards are expected. Further work has incorporated policy summary [21] and abstract action spaces for robotics [12] into the explanations. Alternatively, Tsuchiya *et al.* [27] proposed decomposing the Q-value into the importance of future states and Yau *et al.* [29] into the probabilities of state transitions. However, neither state decomposition proposal has been shown to scale to complex environments where explanations are most important for understanding agents. Importantly, all these decomposition approaches differ from our work as they require decomposing the reward estimator into components or states rather than over time, although future work could explore combining these approaches.

An alternative approach to understanding an agent's rewards is to modify the environment's reward function. Mirachandani and Karamcheti [15] proposed a reward-shaping approach using natural language to convert long-horizon task descriptions to lower-level dense rewards. This allows the higher-level descriptions to be used to explain the agent policy however this relies on correctly interpreting these complex descriptors while our work requires no modification to the environment setup.

For approaches that contribute similar applications to TRD, Madumal *et al.* [13] proposed a text-based explanation using a hand-crafted causal model of a Starcraft 2 agent to explain why (or why not) to take an action with respect to environmental state variables from the causal model. Our explanation similarly illustrates the future reasoning of an agent, but does so in terms of rewards not changes to the environment's state. Most importantly, our approach requires neither a hand-crafted causal model nor explicitly identified environment features. To explain the agent's focus within an observation, Greydanus *et al.* [9] proposed a perturbation-based saliency map that uses the changes in the policy output for noise applied to an area of the observation to understand a region's importance to the agent. This is limited to only visualising a region's importance for all future rewards, whereas combining TRD with saliency map algorithms can explain a particular future reward's regions of importance in decision-making.

Outside XRL, researchers have explored non-scalar variants of the Q-value, primarily for improving performance. Bellemare *et al.* [5] proposed C51, a training algorithm that learns the distribution of cumulative rewards rather than just the expectation, achieving state-of-the-art performance in Atari. Our work differs as we propose decompose the Q-value into the expected reward for future timesteps rather than the probability distribution over all future rewards. Furthermore, we propose new explanatory applications that are facilitated by TRD.

3 Preliminaries

Before we present TRD, we provide sufficient technical detail on methods we build upon: Markov Decision Processes to mathematically describe TRD; Deep Q-learning for learning Q-value functions in complex environments; QDagger for learning with pretrained agents; and GradCAM for creating saliency map explanations.

To model a reinforcement learning environment, we use a Markov Decision Process [17] described by the tuple $\langle S, A, R, P, T\rangle$. These variables denote the set of possible states and actions (S and A respectively), the reward function ($R(s,a)$) given a state action s, a that is bounded to finite values, the transition probability ($P(s'|s,a)$) of the next state (s') given the prior state-action (s,a) and the termination condition ($T(s)$) that returns True if the state (s) is a goal state. For simplicity, following Sutton and Barto [24], we denote S_i, A_i and R_i as the state, action and reward received for timestep i.

Given an environment, we wish to learn a policy π that maximises its cumulative rewards over an episode. Furthermore, to incentivise the agent to collect rewards sooner, we apply an exponential discount factor ($\gamma \in [0,1)$). For a policy, π, we may define the expected sum of future rewards in terms of the Q-value, $q_\pi(s,a)$, or the state-value, $v_\pi(s)$, functions, Eqs. (1) and (2) respectively.

$$q_\pi(s,a) = \mathbb{E}_\pi\Big[\sum_{n=0}^{\infty}\gamma^n R_{t+n}|S_t = s, A_t = a\Big] \tag{1}$$

$$v_\pi(s) = \mathbb{E}_\pi\Big[\sum_{n=0}^{\infty}\gamma^n R_{t+n}|S_t = s\Big] \tag{2}$$

To learn an optimal policy, agents can select actions that maximise the Q-value for a given state. Using this, Watkins and Dayan [28] proposed iteratively minimising the error between the predicted Q-value for a state-action and a bootstrapped target Q-value using the state-action's resultant reward plus the maximum Q-value in the next timestep (Eq. (3)), referred to as Q-learning. Importantly, given initial conditions and an infinite number of iterations, [28] proved Q-learning would converge to the optimal policy.

$$L_{\text{Q}}(D) = \mathbb{E}_{(s,a,R,s')\sim D}(q_\pi(s,a) - y_{\text{target}})^2 \tag{3}$$

$$y_{\text{target}} = R + \gamma \max_{a'\in A}\hat{q}_\pi(s',a') \tag{4}$$

This was extended by Mnih *et al.* [16] to use neural networks, referred to as Deep Q-learning (DQN) for a general RL algorithm that achieved state-of-the-art performance in image-based environments.

They combined several extensions to Q-learning including an experience replay buffer to store training examples, a target network for stability and a convolutional neural network to learn the Q-values.

To help minimise the training time of TRD agents, we utilise QDagger [2], a workflow for learnt policies to reuse or transfer their knowledge to new agents. In particular, QDagger proposes two changes to an agent's training scheme: an offline training stage using a teacher's (pretrained agent) replay buffer and adding a distillation loss to the agent's (student) policy loss that minimises the KL divergence between the student's π and teacher's π_T policy (Eq. (5)). The weighting of this loss term is controlled by λ_T as the ratio of teacher to student average reward. Agarwal *et al.* [2] showed QDagger allows student agents to match the teacher's performance for Atari environments with 20x fewer observations than normal.

$$L_{\text{QDagger}}(D) = L_Q(D) + \lambda_T \mathbb{E}_{s\sim D}\left[\sum_a \pi_T(a|s)\log\pi(a|s)\right] \quad (5)$$

We utilise GradCAM [20], a popular saliency map algorithm highlighting the input features that have the greatest influence on a neural network's decision-making. For a given convolutional layer, GradCAM computes the gradients from the layer's features to one of the network's outputs such that the gradient is proportional to the feature's importance in that network's decision-making for the output.

4 Temporal Reward Decomposition

As described in Section 1, due to the scalar output of future reward estimators (i.e., Q-value and state-value functions), their reward composition cannot be known, preventing understanding when and what future rewards the agent expects to receive. We, therefore, propose a novel future reward estimator (Eqs. (6)), referred to as Temporal Reward Decomposition (TRD) that predicts an agent's next N expected rewards. Furthermore, we prove its equivalence to scalar future reward estimators and provide a bootstrap-based loss function to learn the estimator (Eq. (15)). For consistency, all equations in this section are for the Q-value with state-value equations in Appendix A of the supplementary material [25].

Before defining our TRD-based future reward estimators, to prove their equivalence to scalar future reward estimators (Eq. (7)), we first prove that the expected sum of future rewards is equivalent to the sum of expected future rewards enabling the decomposition of rewards in Eq. (6): Theorem 1.[1]

$$q_\pi^{\text{TRD}}(s,a) = \begin{pmatrix} \mathbb{E}_\pi[R_t|S_t=s,A_t=a] \\ \mathbb{E}_\pi[\gamma R_{t+1}|S_t=s,A_t=a] \\ \vdots \\ \mathbb{E}_\pi[\gamma^{N-1}R_{t+N-1}|S_t=s,A_t=a] \\ \mathbb{E}_\pi\left[\sum_{i=N}^{\infty}\gamma^i R_{t+i}|S_t=s,A_t=a\right] \end{pmatrix} \quad (6)$$

$$\sum q_\pi^{\text{TRD}}(s,a) \equiv q_\pi(s,a) \qquad \forall s\in S, \forall a\in A \quad (7)$$

Using the notation in Section 3, we propose Eq. (6) that outputs a vector of the next N expected rewards with the last element being equal to the cumulative sum of expected rewards from N to ∞. Each element i refers to the expected reward in $t+i$ timesteps with the final element being the sum of rewards beyond N timesteps. Using Theorem 1, Eq. (6) is provably equivalent to the scalar Q-value by summing over the array elements (Eq. (7)) through expanding Eq. (11) with $N+1$ expectations. Critically, this equivalence is not reversible such that given a scalar Q-value, Eq. (6) cannot be known.

Theorem 1. *Given a state s and action a, the expected sum of rewards is equal to the sum of expected rewards, more precisely $\mathbb{E}_\pi\left[\sum_{i=0}^{\infty}\gamma^i R_{t+i}|S_t=s,A_t=a\right] \equiv \sum_{i=0}^{\infty}\mathbb{E}_\pi[\gamma^i R_{t+i}|S_t=s,A_t=a]$ for all $s\in S$ and $a\in A$.*

Proof.

$$\mathbb{E}_\pi\left[\sum_{i=0}^{\infty}\gamma^i R_{t+i}\Big|S_t=s,A_t=a\right] \quad (8)$$

$$=\mathbb{E}_\pi\left[R_t+\sum_{i=1}^{\infty}\gamma^i R_{t+i}\Big|S_t=s,A_t=a\right] \quad (9)$$

$$=\mathbb{E}_\pi[R_t|S_t=s,A_t=a] + \mathbb{E}_\pi\left[\sum_{i=1}^{\infty}\gamma^i R_{t+i}\Big|S_t=s,A_t=a\right] \text{ (given LoE}^1\text{)} \quad (10)$$

$$=\mathbb{E}_\pi[R_t|S_t=s,A_t=a] + \mathbb{E}_\pi[\gamma R_{t+1}|S_t=s,A_t=a] + \mathbb{E}_\pi\left[\sum_{i=2}^{\infty}\gamma^i R_{t+i}\Big|S_t=s,A_t=a\right] \quad (11)$$

$$=\sum_{i=0}^{\infty}\mathbb{E}_\pi[\gamma^i R_{t+i}|S_t=s,A_t=a] \quad (12)$$

□

Implementing TRD within a deep RL agent's future reward estimator requires two primary changes. The first is increasing the neural network output by $N+1$; i.e., the size of Eq. (6) for predicting the next N future rewards. The second is the loss function (Eq. (15)) for the network to learn Eq. (6). Additionally, as the network now outputs a vector of future rewards rather than a scalar, for action selection and other applications, q_π can be recovered by summing across vector elements before being applied as normal (Eq. (7)). Appendix B [25] includes pseudocode for implementing the loss function, and the associated GitHub repository[2] contains the implementation of a TRD-modified DQN training algorithm using Gymnasium [26].

For long-horizon environments where an agent may take hundreds or thousands of actions, TRD is limited in scale as the number of predicted rewards scales linearly with the number of output neurons. We therefore propose an alternative approach to preserve the temporal distance that can be explained using a fixed number of output neurons. Rather than each vector element predicting an individual reward, Eq. (13) groups rewards in each vector element; e.g., for pair grouping $[R_t+R_{t+1}, R_{t+2}+R_{t+3}, \dots]$. This approach, denoted w for the reward grouping size, scales linearly with the number of future rewards by w for a fixed N such that the total number of predicted rewards is $N\cdot w$. Importantly, like Eq. (6), Eq. (13) is equivalent to the Q-value by summing across elements (Eq. (14)) using $N\cdot w+1$ expansions of Eq. (11). Additionally, for $w=1$, Eq. (13) is equivalent to Eq. (6) and implementation only requires utilising an N-step [24] experience replay buffer to compute the sum of the first w rewards and the next observation in w timesteps.

As a result, N and w present a trade-off between the reward vector size (N) and precise knowledge of each timestep's expected reward

[1] Linearity of Expectation (LoE) is a property that any expectation can be split into its linear components, even for *dependent* random variables [23, Page 166].

[2] https://github.com/pseudo-rnd-thoughts/temporal-reward-decomposition

$$q_\pi^{\text{TRD}}(s,a) = \begin{pmatrix} \mathbb{E}_\pi[R_t|S_t=s,A_t=a] + \cdots + \mathbb{E}_\pi[\gamma^{w-1}R_{t+w-1}|S_t=s,A_t=a] \\ \mathbb{E}_\pi[\gamma^{w}R_{t+w}|S_t=s,A_t=a] + \cdots + \mathbb{E}_\pi[\gamma^{2w-1}R_{t+2w-1}|S_t=s,A_t=a] \\ \vdots \\ \sum_{i=(N-1)w}^{Nw}\mathbb{E}_\pi[\gamma^i R_{t+i}|S_t=s,A_t=a] \\ \mathbb{E}_\pi[\sum_{i=Nw}^{\infty}\gamma^i R_{t+i}|S_t=s,A_t=a] \end{pmatrix} \tag{13}$$

$$\sum q_\pi^{\text{TRD}}(s,a) \equiv q_\pi(s,a) \qquad \forall s \in S, \forall a \in A \tag{14}$$

$$L_{\text{TRD}} = \mathbb{E}_{(s_t,a_t,R_{t+i},s_{t+w})\sim D}\begin{bmatrix} \left(q_\pi^{\text{TRD}_0}(s_t,a_t) - \sum_{i=0}^{w} R_{t+i}\right)^2 \\ \left(q_\pi^{\text{TRD}_1}(s_t,a_t) - \gamma^w q_\pi^{\text{TRD}_0}(s_{t+w},a')\right)^2 \\ \left(q_\pi^{\text{TRD}_2}(s_t,a_t) - \gamma^w q_\pi^{\text{TRD}_1}(s_{t+w},a')\right)^2 \\ \vdots \\ \left(q_\pi^{\text{TRD}_N}(s_t,a_t) - \gamma^w q_\pi^{\text{TRD}_{N-1}}(s_{t+w},a')\right)^2 \\ \left(q_\pi^{\text{TRD}_{N+1}}(s_t,a_t) - \gamma^w (q_\pi^{\text{TRD}_N}(s_{t+w},a') + q_\pi^{\text{TRD}_{N+1}}(s_{t+w},a'))\right)^2 \end{bmatrix} \tag{15}$$

(w). For example using Figure 1, if $w=2$ and $N=2$ then the q_π^{TRD} for moving up is $[0,0,3.26]$ as $[0+0,0+0,3.26]$ and moving down is $[0.95,1.76,0.81]$ as $[0+0.95,0.90+0.86,0.81]$. Furthermore, to predict, for example, the next 30 rewards, $N=30, w=1$ and $N=6, w=5$ are both valid parameters. We explore the impact of these parameters on training in Section 5.

Through experiment, we found that converting q_π^{TRD} to q_π by summing over elements (Eq. (7)), then using the scalar loss function (Eq. (3)) does not converge to q_π^{TRD}. Therefore, based on the Q-learning loss function (Eq. (3)), we define a novel element-wise mean squared error of reward vectors (Eq. (15)) where a' denotes the optimal next action ($\arg\max_{a\in A}\sum q_\pi^{\text{TRD}}(s_{t+w},a)$) and we use the following notation to index an element of the reward vector:

$$q_\pi^{\text{TRD}_0}(s,a) = \mathbb{E}_\pi[R_t|S_t=s,A_t=a] \tag{16}$$

$$q_\pi^{\text{TRD}_1}(s,a) = \mathbb{E}_\pi[\gamma R_{t+1}|S_t=s,A_t=a] \tag{17}$$

$$q_\pi^{\text{TRD}_N}(s,a) = \mathbb{E}_\pi[\gamma^{N-1}R_{t+N-1}|S_t=s,A_t=a] \tag{18}$$

$$q_\pi^{\text{TRD}_{N+1}}(s,a) = \mathbb{E}_\pi\left[\sum_{i=N}^{\infty}\gamma^i R_{t+i}|S_t=s,A_t=a\right] \tag{19}$$

For Eq. (15), we construct a predicted and bootstrap-based target value (cf. Q-learning), computing the element-wise mean squared error of the predicted and target reward vectors. The prediction is the reward vector for the action taken in state t, $q_\pi^{\text{TRD}}(s_t,a_t)$. For the target, the first element is the actual reward collected (R_t to R_{t+w}) followed by the reward vector for the optimal action in s_{t+w}, $q_\pi^{\text{TRD}}(s_{t+w},a')$, shifted along one position with the last two elements combined. We do this because element i of the reward vector, $q_\pi^{\text{TRD}_i}(s_t,a_t)$, refers to the predicted reward in $t{+}i$ timesteps, for the next observation, $t{+}w$, the equivalent reward vector element is $i{-}1$ in the target vector, $\gamma q_\pi^{\text{TRD}_{i-1}}(s_{t+w},a')$.

5 Retraining Pretrained Agents for TRD

The goal of Temporal Reward Decomposition (TRD) is to provide information about an agent's expected future rewards over time so that we can use this information to better understand its behaviour. For this to be practically effective, TRD agents should be capable of achieving performance similar to their associated base RL agent. In this section, therefore, we evaluate the performance of DQN agents [16] that incorporate TRD for a range of Atari environments [4] and assess the impact of TRD's two hyperparameters on training: reward vector size, N; and reward grouping, w.

We conduct hyperparameter sweeps across each independently, varying N, w, and $N\cdot w$, across three Atari environments (Breakout, Space Invaders and Ms. Pacman), each containing different reward functions. To account for variability in training, we repeat our hyperparameter sweeps three times. The training hyperparameters and hardware used in training, along with the agent's final scores, are presented in Appendix B [25]. Training scripts and final neural network weights for all runs are provided in the associated GitHub repository.

Rather than training agents from scratch for these environments, we use open-sourced pretrained Atari agents [10] and the QDagger training workflow [2], described in Section 3.

Using periodic evaluation on the same ten seeds, Figure 2 plots the teacher normalised interquartile mean [1] of the episodic reward. We find that all three hyperparameter sweeps enable the agent to approach the pretrained (teacher) agent's score with neither parameter having a significant detrimental impact. Only the offline training for a constant temporal distance ($N\cdot w=24$) do the agents with smaller values of w showcase greater initial performance, but this difference is resolved during the online training stage.

Interestingly, for the sweep of N, we found no degradation in performance, which was unexpected as we believed that larger values of N would require more training to reach the same performance. As a result, in Section 6, we trained agents with $N{=}40, w{=}1$. Further work is required to understand if these performance curves hold for larger values of N and for more complex environments or agents.

To verify that our TRD loss function (Eq. (15)) converges to a policy that is similar to the pretrained agent's scalar Q-value. Figure 3 plots the mean squared error of the Q-values for both pretrained DQN agents and TRD agents during training. We find all parameters get close to the pretrained agent's Q-value with $w{=}1$ being an important factor.

Regarding the computation impact of incorporating TRD, we found that our QDagger+TRD DQN agents took $\approx 10\%$ fewer steps per second than our base DQN agents, 248 to 274 steps per second, respectively. This performance will be jointly caused by QDagger requiring an additional forward pass from the teacher agent and TRD

Figure 2. Interquantile mean training curves for Atari TRD-DQN agents for three environments (Breakout, Space Invaders and Ms Pacman) with three repeats, normalised by the teacher's score. Offline and Online indicate where training used the offline replay buffer and the online environment steps.

Figure 3. The Mean Squared Error between the student (TRD agent) and pretrained teacher agent averaged over three Atari environments with three repeats. Offline and Online indicate where training used the offline replay buffer and the online environment steps.

using a larger network output and a more complex loss function.

6 Explaining an Agent's Future Beliefs and Decision-Making

We now present three novel explanation mechanisms using future expected rewards: understanding what rewards the agent expects to receive and when, and their confidence in this prediction; visualising an observation feature's importance for predicting rewards at near and far timesteps; and a contrastive explanation using the difference in future rewards to understand the impact of different actions choices (Sections 6.1, 6.2, and 6.3 respectively). We showcase these applications using three different Atari environments with more examples in Appendices C, D, and E of the supplementary material [25]. All agents were retrained DQN agents incorporating TRD using $N{=}40$ and $w{=}1$.

6.1 What Rewards to Expect and When?

For environments with complex transition dynamics or reward functions such as Atari, understanding how an agent maximises its future rewards or predicting what rewards it will receive and when is not possible, unlike with the toy example illustrated in Figure 1. We show here how a TRD agent's predicted future rewards supply this information, presenting an important explanatory perspective for understanding agent decisions. Furthermore, for environments with binary reward functions (i.e., where the rewards are either zero or a constant value) the agent's expected reward can be further decomposed into the probability of the reward function components. Atari uses integer rewards and DQN agents clip rewards to -1 to 1, and so for these examples the agent's probability of collecting a reward is equivalent to the reward's expectation.

Figures 4 and 5 plot the agent's expected rewards over the next 40 timesteps for the observation on the left. As $w{=}1$, the discount factor is constant for each predicted timestep, and so we factor it out, leaving just the expected reward. Without domain knowledge of each environment and its reward functions, we can observe from the expected rewards plots that the agent expects periodic non-zero rewards every 8 to 9 timesteps for Space invaders and every 15 timesteps for Breakout. Additionally, considering that the expected rewards (for these environments) are equivalent to the agent's confidence (probability) in receiving a reward for a particular timestep, users can infer that the agent's confidence reduces over time for the specific timestep that the agent will receive a reward. As such, for space invaders, the agent has high confidence for the close timesteps ($t{+}6$ and $t{+}15$) with the expected rewards for the third and fourth rewards being distributed across several timesteps ($t{+}23$ to $t{+}24$ and $t{+}30$ to $t{+}40$).

Further, utilising domain knowledge of each environment, Figures 4 and 5 correlate with our understanding as agents can only

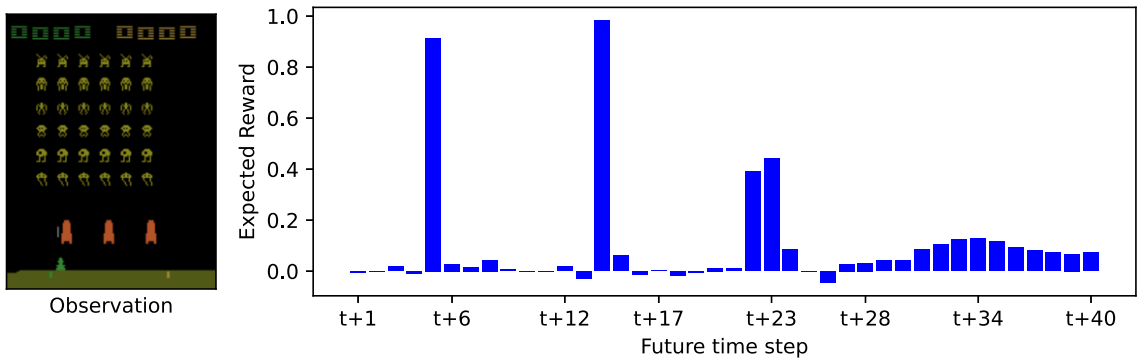

Figure 4. A Space Invaders observation (left) with the respective predicted next 40 future expected rewards (right).

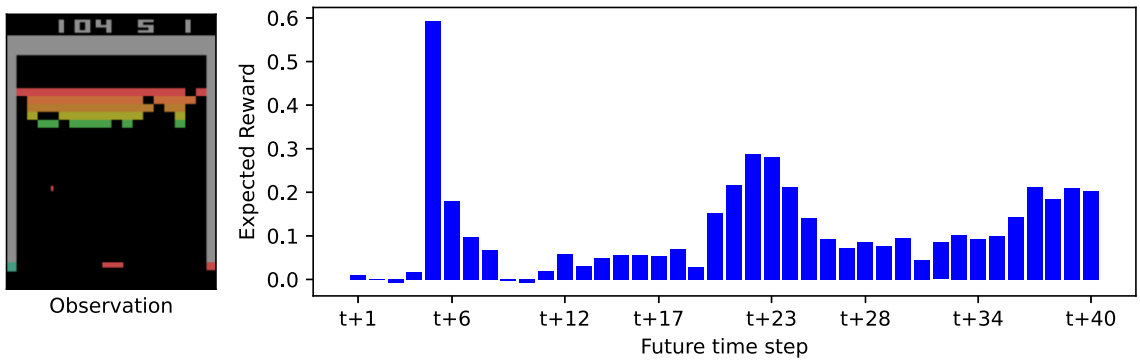

Figure 5. A Breakout observation (left) with the respective predicted next 40 future expected rewards (right).

shoot aliens or break bricks periodically. Additionally, as the policy is stochastic due to epsilon-greedy action selection and with randomness in the environment, the uncertainty of timesteps far in the future is unsurprising and matches with human expectations.

Building on the two figures, we can generate videos of the agent's expected rewards across entire episodes plotting the expected reward for each observation. Example videos are provided in the associated GitHub repository and contain significant additional context for users to visualise how the agent's predicted future rewards change over time as the environment's state evolves.

As a result, we anticipate that TRD has the potential to aid researchers and developers debug RL agents; Figure 4 and the related videos provide novel information about an agent's future beliefs and its understanding of an environment's reward function.

6.2 *What Observation Features are Important?*

Understanding the areas of an input that have the greatest impact on a neural network is a popular technique for generating explanations, called saliency maps. These allow users to visualise what features of an observation most influence an agent's decision. With access to an agent's beliefs about its future expected rewards, TRD provides novel saliency map opportunities to understand how the agent's focus with respect to an observation varies.

Utilising GradCAM [20] (a popular saliency map algorithm described in Section 3), we can select individual expected rewards as the output to discover its feature importance. Figure 6 plots an Atari Breakout observation and the normalised feature importance for the expected reward of the next timestep ($t+1$) and the most distance expected reward ($t+40$) along with their normalised difference. The feature importance plots highlight areas of focus (red), influencing its decision and ignored areas (blue). We find that the agent's focus on the ball and bricks vary depending on how far in the future a reward is predicted. For the $t+1$ feature importance, the agent is highly focused (shown in red) on the ball in the centre. In comparison, for $t+40$, the agent has a greater focus on the bricks than the ball. Using domain knowledge of the environment validates human expectations as the number of bricks left and their position will have greater long-term importance to the agent than the ball. This difference is highlighted when subtracting the feature importance of $t+1$ from $t+40$ such that the ball's importance is significantly lower (shown in blue) and the bricks have relatively greater importance (shown in red).

To help visualise this change in an observation feature's importance across each predicted future reward, we provide a video of Figure 6 within the associated GitHub repository. Additionally, we provide a video of an episode plotting the first and last predicted reward's feature importance for each timestep. Like visualising an agent's expected reward in Section 6.1, Figure 6 and videos can help researchers and developers understand in what context a feature has importance for an agent. Previously, it was only possible to understand a feature's importance to predict the agent's total reward, whereas TRD provides us with the ability to investigate the importance of features in a more granular way.

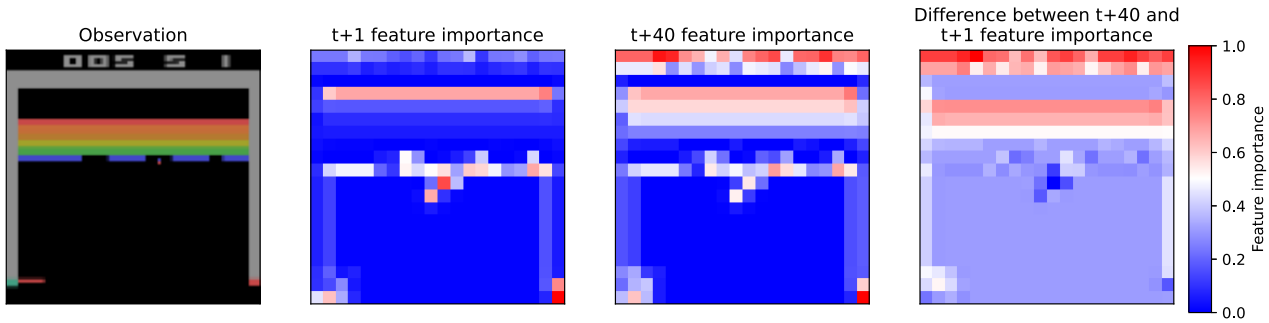

Figure 6. GradCAM saliency maps for the $t+1$ and $t+40$ expected reward along with their difference for a Breakout observation. GradCAM uses the first convolutional layer of the agent's neural network to differentiate.

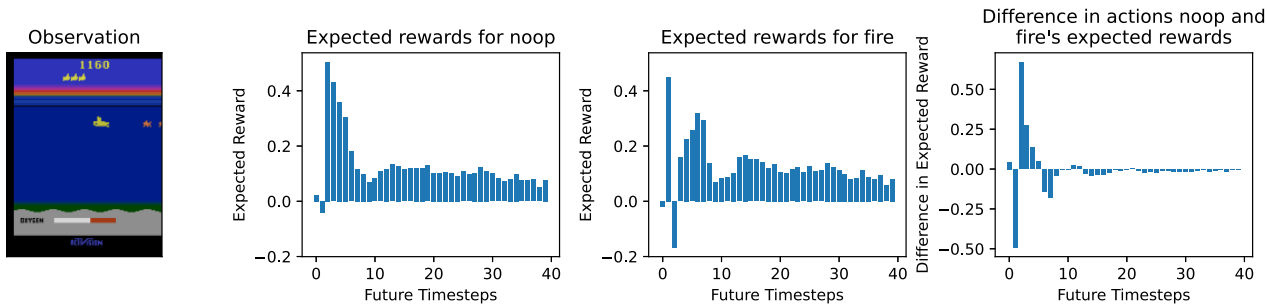

Figure 7. The difference of each future expected reward for taking Left and Right actions of the observation for the Atari Seaquest environment.

6.3 What is the Impact of an Action Choice?

Within an environment, there are often multiple (possibly similar) paths to complete a goal with humans interested in understanding the differences between them (e.g., Figure 1). Contrastive explanations are a popular approach to understanding the reasons for taking one decision over another. In our case, this is the choice between two alternative actions in some state [14]. With the future expected rewards, TRD provides additional information to compare and contrast states and actions using what rewards the agent expects to receive and when along different paths. In this section, we show how simple explanations only using the timestep-wise difference in expected rewards can help understand an action's impact on an agent's future rewards.

Figure 7 shows the expected reward for taking no action (noop) and firing and the differences between the expected reward for noop and firing in the Atari Seaquest environment. The right-hand side figure shows that the difference in future rewards produces a positive and negative spike after which the expected rewards converge. We can infer from these spikes that if the agent fires rather than noop then there is a more immediate reward, whereas if the agent waits, taking no action, the reward is delayed resulting in a later spike. Crucially, this difference in reward outcomes is resolved afterwards causing no long-term difference in the agent's expected rewards. Using domain knowledge, we can assume that this means if the agent doesn't fire in this timestep, it will most likely fire in the following timestep or soon after, thus receiving a slightly delayed reward.

Collectively, with the explanations from Sections 6.1 and 6.2, contrastive explanations highlight the consequences of different actions on an agent's future rewards.

7 Conclusion

Temporal Reward Decomposition (TRD) is a novel reward estimator is equivalent to scalar future reward estimators that can uniquely reveal additional information about deep reinforcement learning agents. We have shown that pretrained Atari agents can be efficiently retrained to incorporate TRD with minimal impact on performance. Furthermore, we have showcased three novel explanatory mechanisms enabled by TRD, demonstrating how these can aid in researchers' and developers' understanding of agent behaviour for complex environments such as the three Atari environments considered here. We can ask "What rewards does an agent expect and when?" by predicting rewards numerous timesteps into the future and the confidence the agent has in their prediction (Section 6.1). We can ask "What observation features are important?" by revealing how an agent's focus changes depending on the immediacy of the reward predicted (Section 6.2). Lastly, we can ask "What is the impact of an action choice?" by revealing the difference in future expected rewards for two alternative actions (Section 6.3).

TRD can be extended in various ways to better explain an agent's future rewards. Incorporating prior decomposition approaches such as Juozapaitis *et al.* [11] to explain the future expected rewards of different reward components is a clear option for future research. Further, Section 6.1 analysis of agent confidence was limited to environments with binary reward functions. Future work could explore how each reward could be modelled as a probability distribution across reward values [5]. This would enable a more detailed understanding of the agent's expectations/confidence in individual future rewards. A further avenue for future research is that the linear relationship between future reward estimators and TRD (Eq. (7)) may be exploited for more efficient training of pre-trained agents.

Acknowledgements

This work was supported by the UKRI Centre for Doctoral Training in Machine Intelligence for Nano-electronic Devices and Systems [EP/S024298/1] and RBC Wealth Management.

Thanks to John Birbeck for advice on the proof of Theorem 1.

References

[1] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

[2] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare. Reincarnating reinforcement learning: Reusing prior computation to accelerate progress. *Advances in Neural Information Processing Systems*, 35:28955–28971, 2022.

[3] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR, 2020.

[4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[5] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[6] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[7] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. Challenges of real-world reinforcement learning: Definitions, benchmarks and analysis. *Machine Learning*, 110(9): 2419–2468, 2021.

[8] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[9] S. Greydanus, A. Koul, J. Dodge, and A. Fern. Visualizing and understanding atari agents. In *International conference on machine learning*, pages 1792–1801. PMLR, 2018.

[10] S. Huang, Q. Gallouédec, F. Felten, A. Raffin, R. F. J. Dossa, Y. Zhao, R. Sullivan, V. Makoviychuk, D. Makoviichuk, M. H. Danesh, C. Roumégous, J. Weng, C. Chen, M. M. Rahman, J. G. M. Araújo, G. Quan, D. Tan, T. Klein, R. Charakorn, M. Towers, Y. Berthelot, K. Mehta, D. Chakraborty, A. KG, V. Charraut, C. Ye, Z. Liu, L. N. Alegre, A. Nikulin, X. Hu, T. Liu, J. Choi, and B. Yi. Open RL Benchmark: Comprehensive Tracked Experiments for Reinforcement Learning. *arXiv preprint arXiv:2402.03046*, 2024. URL https://arxiv.org/abs/2402.03046.

[11] 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.

[12] W. Lu, X. Zhao, S. Magg, M. Gromniak, M. Li, and S. Wermterl. A closer look at reward decomposition for high-level robotic explanations. In *2023 IEEE International Conference on Development and Learning (ICDL)*, pages 429–436. IEEE, 2023.

[13] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere. Explainable reinforcement learning through a causal lens. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 34, pages 2493–2500, 2020.

[14] T. Miller. Contrastive explanation: A structural-model approach. *The Knowledge Engineering Review*, 36:e14, 2021.

[15] S. Mirchandani, S. Karamcheti, and D. Sadigh. Ella: Exploration through learned language abstraction. *Advances in neural information processing systems*, 34:29529–29540, 2021.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

[17] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[18] Y. Qing, S. Liu, J. Song, and M. Song. A survey on explainable reinforcement learning: Concepts, algorithms, challenges. *arXiv preprint arXiv:2211.06665*, 2022.

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

[20] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017.

[21] Y. Septon, T. Huber, E. André, and O. Amir. Integrating policy summaries with reward decomposition for explaining reinforcement learning agents. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 320–332. Springer, 2023.

[22] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[23] D. Stirzaker. *Elementary Probability*. Cambridge University Press, 2003.

[24] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[25] M. Towers, Y. Du, C. Freeman, and T. J. Norman. Explaining an agent's future beliefs through temporally decomposing future reward estimators. *arXiv preprint arXiv:2408.08230*, 2024. Full version of this paper.

[26] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

[27] Y. Tsuchiya, Y. Mori, and M. Egi. Explainable reinforcement learning based on q-value decomposition by expected state transitions. *CEUR Workshop Proceedings*, 2023.

[28] C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[29] 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*, 33:18375–18386, 2020.