Explainable Reinforcement Learning via Reward Decomposition

Zoe Juozapaitis[1] , **Anurag Koul**[1] , **Alan Fern**[1] , **Martin Erwig**[1] , **Finale Doshi-Velez**[2]
[1]**Oregon State University**
[2]**Harvard**
juozapaz@oregonstate.edu, koula@oregonstate.edu, Alan.Fern@oregonstate.edu,
Martin.Erwig@oregonstate.edu, finale@seas.harvard.edu,

Abstract

We study reward decomposition for explaining the decisions of reinforcement learning (RL) agents. The approach decomposes rewards into sums of semantically meaningful reward types, so that actions can be compared in terms of trade-offs among the types. In particular, we introduce the concept of minimum sufficient explanations for compactly explaining why one action is preferred over another in terms of the types. Many prior RL algorithms for decomposed rewards produced inconsistent decomposed values, which can be ill-suited to explanation. We exploit an off-policy variant of Q-learning that provably converges to an optimal policy and the correct decomposed action values. We illustrate the approach in a number of domains, showing its utility for explanations.

1 Introduction

Many RL methods estimate Q-values in order to evaluate and select actions. Q-values, however, usually do not give insight into factors contributing to action choices. For instance, in a racing game it may not be clear whether the agent swerved to get closer to the goal (increasing reward), or to avoid an obstacle (avoiding a crashing penalty). In this paper, we explore *reward decomposition* for gaining such insight, where an environment's reward is decomposed into a sum of meaningful reward types. Many environments have natural reward decompositions, yet they are not made explicit to RL agents. Our key idea is to expose these decompositions to RL agents, which can be leveraged to provide explanations for decisions in terms of trade-offs among the reward types.

Prior work has considered RL with reward decomposition, but with a focus on expediting learning, rather than explanation. We show that this prior work has fundamental problems, which limits their explanation utility. To address this, we adapt a prior off-policy multi-agent RL algorithm, with unclear convergence guarantees, to our decomposed reward setting. We give the first proof that, in the table-based setting, this approach converges to the correct decomposed Q-functions, thus, supporting accurate explanations. We also introduce a DQN-variant to utilize function approximation.

In addition, we propose the concept of *minimal sufficient explanation* (MSX) for compactly explaining action preferences via decomposed rewards. These explanations are integrated into a domain-independent "explanation interface", which will be made publicly available along with the corresponding RL algorithms. Our experiments focus on demonstrating the interface's potential utility to RL practitioners via three case studies. The results show that significant insights about an agent's behavior can be gained, e.g. explaining strange behavior, identifying "bugs" in preferences, identifying the influence of shaping rewards, and helping to identify issues with the numerical optimizer.

2 Reward Decomposition

A Markov Decision Processes (MDPs) is a tuple (S, A, T, R), where S and A are finite sets of states and actions, $T(s, a, s')$ is the probability of transitioning to state s' after taking action a in s, and $R(s, a)$ is the reward for taking a in s. A policy $\pi(s)$ returns an action to take in state s and the associated Q-function, $Q^\pi(s, a)$, gives the expected infinite-horizon γ-discounted cumulative reward of taking action a in state s and following π thereafter. $Q^*(s, a)$ denotes the Q-function of the optimal policy π^*, which satisfies $\pi^*(s) = \arg\max_{a \in A} Q^*(s, a)$.

We focus on explanations for RL agents that learn Q-functions, which allow for observing how much an agent prefers one action over another. Raw Q-values, however, give no insight into the positive and negative factors contributing to the preferences. For example, the Dota 2 RL domain [12] uses a reward function based on the sum of over 10 reward types measuring quantities such as net worth, kills, deaths, assists, last hits, etc. Explaining action preferences in terms of such meaningful types could provide significant insight. Typical RL settings, however, do not support such explanations, since the individual reward types are mixed as a lump-sum scalar reward. To support such explanations, we explicitly expose the types to the agent via *reward decomposition*.

Algorithm 1 Table-Based Decomposed Reward RL: The pseudo-code can be instantiated for drQ, HRA, and drSARSA by using the specified assignment to a'. The notation $Q \overset{\alpha}{\leftarrow} x$ is shorthand for $Q \leftarrow (1-\alpha)Q+\alpha x$. ϵ-greedy exploration is used, but could be replaced with other mechanisms.

$s_0 \leftarrow$ Initial State
$a_0 \leftarrow \epsilon(Q^0, s_0)$
$t = 0$
repeat
  $(s_{t+1}, \vec{r}_t) \leftarrow \text{Act}(a_t)$
  $a_{t+1} \leftarrow \epsilon_t(Q^t, s_{t+1})$;; ϵ-greedy exploration
  for all $c \in C$ **do**

$$a' \leftarrow \begin{cases} \arg\max_a \sum_c Q_c^t(s,a) & \text{drQ} \\ \arg\max_a Q_c(s,a) & \text{HRA} \\ a_{t+1} & \text{drSARSA} \end{cases}$$

  $Q_c^{t+1}(s_t, a_t) \overset{\alpha_t}{\leftarrow} r_{t,c} + \gamma Q_c^t(s_{t+1}, a')$
  $t \leftarrow t+1$
  end for
until convergence

The MDP formulation can incorporate reward decomposition by specifying a set of *reward components/types* C and defining a vector-valued reward function $\vec{R} : S \times A \rightarrow \mathbb{R}^{|C|}$, where $R_c(s,a)$ is the reward for type $c \in C$. The objective is still to optimize the overall (mixed) reward function $R(s,a) = \sum_{c \in C} R_c(s,a)$. However, the vector-valued reward allows for defining a vector-valued Q-function $\vec{Q}^\pi$, where $Q_c^\pi(s,a)$ gives action values that account for only rewards type c. These definitions imply that the overall Q-function also decomposed as $Q^\pi(s,a) = \sum_c Q_c^\pi(s,a)$. Below we describe how to learn such decomposed Q-functions and then use them for explanation. Note that, our notion of reward decomposition is orthogonal and complementary to temporal reward decomposition, which is commonly studied in hierarchical reinforcement learning (e.g. [7; 15; 3; 4; 14]).

3 RL with Reward Decomposition

Recent work [16] considered reward decomposition for speeding up RL. Earlier work also considered analogs of reward decomposition for multi-agent RL [5; 10]. No prior work, however, has studied reward decomposition for our primary purpose of explanation. This perspective illuminates certain practical and theoretical deficiencies of prior methods. Below we describe the methods used in our experiments and fill a significant theoretical hole that is particularly relevant for explanation.

HRA [16]. Algorithm 1 shows that (table-based) HRA updates each component Q_c independently via standard Q-learning updates [17]. Thus, each Q_c converges to the optimal Q-function with respect to reward type c. HRA then uses the sum of components $Q_{\text{HRA}}(s,a) = \sum_{c \in C} Q_c(s,a)$ as an estimate of the optimal Q-function. This heuristic showed success in two domains, but generally overestimates the true Q-values, and the corresponding greedy policy can perform arbitrarily poorly, even with infinite training. Even when $Q_{\text{HRA}}(s,a)$ performs well, the components Q_c do not reflect the actual Q-function of the greedy HRA policy, which can make explanations quite unintuitive.

drSARSA. Prior work [5] studied rewards that decompose across multiple agents, which is analogous to our setting with agents corresponding to reward types. An on-policy SARSA variant was proposed, which is identical to standard SARSA [13], but updates each Q-function component Q_c for each experience tuple rather than just the overall Q-function. Algorithm 1 gives a table-based version of drSARSA using ϵ-greedy exploration. Standard SARSA results imply convergence to an ϵ-optimal policy, and each Q_c converges to the correct component Q-function of that policy [5].

Typically, policies learned via SARSA or drSARSA are executed greedily at test time, i.e. without exploration. The Q-components Q_c, however, do not reflect the value of the greedy policy, but rather the value of the exploration policy followed during learning. As our experiments show, this can yield explanations that violate intuition. This can't be fixed by just decreasing ϵ, since our explanations need accurate values for non-greedy actions, which requires non-trivial exploration during learning.

Decomposed Reward Q-Learning (drQ). Neither drSARSA nor HRA are satisfactory approaches for explanation based on reward decomposition. drSARSA is an on-policy algorithm and thus learns the Q-function of an exploration policy instead of the greedy policy. Rather, HRA is an off-policy algorithm, but is unsound. To the best of our knowledge there is no off-policy RL algorithm for decomposed rewards that has been proven to be convergent and sound.

It turns out that a prior off-policy variant of Q-learning for multi-agent RL [10] can serve our purpose when we interpret agents as reward types. The issues of convergence and soundness, however, have not yet been addressed. Algorithm 1 gives this algorithm, referred to as *decomposed reward Q-learning (drQ)*.

Given experience tuple $(s, a, \vec{r}, s')$, drQ updates each component Q_c, like HRA, but with an important difference. HRA adjusts the value of each Q_c toward $r_c + \gamma \max_{a'} Q_c(s', a')$, which completely ignores the influence of other components to the overall Q-function and greedy action. Rather, drQ first computes the overall greedy action, i.e., $a^+ = \arg\max_{a'} \sum_{c \in C} Q_c(s', a')$, and then updates each component $Q_c(s,a)$ toward $r_c + \gamma Q_c(s', a^+)$. Intuitively, this leads Q_c to converge toward the value (w.r.t. c) of the overall greedy policy. Below we show that this approach converges to both the overall optimal policy and the correct component Q-functions.

Convergence of drQ. Let Q_c^t denote the learned Q-function for component c and after t learning updates and $Q^t(s,a) = \sum_{c \in C} Q_c^t(s,a)$ the overall Q-function with $\pi^t(s) = \arg\max_a Q^t(s,a)$ the greedy policy. These

functions are random variables due to randomness in the environment and exploration. We denote the Q-function of the optimal policy with respect to reward type c as $Q_c^{\pi^*}$.

drQ achieves two notions of convergence: (1) convergence of $Q^t(s,a)$ to $Q^*(s,a)$, and (2) convergence of each $Q_c^t(s,a)$ to $Q_c^{\pi^*}(s,a)$. Condition (2) is important for explanations, since it guarantees that we will converge to accurate component Q-functions and, in turn, accurate explanations of π^*.

Standard convergence proofs for Q-learning (e.g. [6; 1]) still apply to (1), but do not directly extend to cover (2). The key difficulty is that standard proofs leverage the fact that Q-learning updates correspond to deterministic operators that are contraction mappings. Rather, the update of drQ does not appear to have such a mapping. However, we are able to show that while there is no such mapping early in learning, such a mapping does exist after a finite number of drQ updates. Further, this mapping has a fixed point equal to the correct component Q-values. Accordingly, the the component Q-functions will converge to the correct values.

Theorem 1. *If drQ is run under the standard conditions for the almost sure (a.s.) convergence of Q-learning,*[1] *then for all c, s, and a, the random variable $Q_c^t(s,a)$ has a.s. converge to $Q_c^{\pi^*}(s,a)$.*

Full proof available in the full paper. The theorem implies that drQ's overall Q-function $Q^t(s,a)$ converges to the optimal Q-function $Q^*(s,a) = \sum_{c \in C} Q_c^{\pi^*}(s,a)$ and hence an optimal policy π^*.

Function Approximation. Our table-based analysis of drQ indicates that the update rule is consistent for decomposed rewards. To support RL environments with enormous state-spaces we extend drQ to function approximation. Specifically, we describe a straightforward extension of DQN [11], for deep RL, which we call *decomposed reward DQN (drDQN)*.

We will assume that each component Q-function is represented by a function approximator $Q_c(s,a;\theta_c)$, where θ are the parameters. For example, Q_c may correspond to a neural network, possibly with the networks for different reward types sharing initial network layers. Like DQN, drDQN stores two sets of parameters, θ_c the current parameters being updated, and θ'_c the "target parameters" used for future value estimates. drDQN operates exactly the same as DQN, except that each component Q-function is updated based on the current greedy action of the target network. In particular, each new experience tuple is added to the replay memory and then a mini-batch of experience tuples $\{(s_i, a_i, \vec{r}_i, s'_i) \,:\, i = 1, \ldots, k\}$ is sampled from the memory. The parameters of each component θ_c are then updated using gradient descent on the loss function shown below:

$$L(\theta_c) = \sum_{i=1}^{k} (y_{c,i} - Q_c(s_i, a_i; \theta_c))^2$$

$$y_{c,i} = \begin{cases} r_c, & \text{for terminal } s'_i \\ r_c + \gamma Q_c(s'_i, a_i^+; \theta'_c), & \text{for non-terminal } s'_i \end{cases}$$

$$a_i^+ = \arg\max_{a'} \sum_{c \in C} Q_c(s'_i, a', \theta'_c)$$

Periodically, the target parameters are replaced with the current learning parameters.

We use this same framework to get function-approximation versions of drSARSA and HRA. We refer to this (Deep) version of drSARSA as *drDSARSA*. The only change from drDQN is to redefine the above loss so that a_i^+ is equal to an action sampled from the ϵ-greedy policy for s'_i. This attempts to simulate the behavior of drSARSA within the memory buffer setting of DQN. In our experience, this approach works as well or better than a traditional SARSA implementation that would update only along the current trajectory. HRA is implemented by defining a_i^+ for each component c such that $a_i^+ = \arg\max_{a'} Q_c(s'_i, a', \theta'_c)$.

4 Decomposition for Explanation

Given the learned decomposed Q-function components Q_c, we now consider how to use them for effective explanations. In particular, we focus on pairwise action explanations where the goal is to explain why one action is preferred to another in a particular state.

Reward Difference Explanations. To gain insight into why an agent prefers action a_1 over a_2 in state s, i.e. $Q(s,a_1) > Q(s,a_2)$, we define the *reward difference explanation (RDX)* as the difference of the decomposed Q-vectors $\Delta(s,a_1,a_2) = \vec{Q}(s,a_1) - \vec{Q}(s,a_2)$. Each component $\Delta_c(s,a_1,a_2)$ of the RDX is a positive or negative *reasons* for the preference depending on whether a_1 has an advantage (disadvantage) over a_2 with respect to reward type c. An RDX can be visualized as a bar chart with one bar for each reason (e.g. see Figure 4).

Minimal Sufficient Explanations. The RDX gives insight into action preferences, but can overwhelm a human when there are many reward types (and hence reasons). To help identify a small set of the most important reasons we introduce the *minimal sufficient explanation (MSX)*. An MSX for a_1 and a_2 in state s is a tuple $\left(\text{MSX}^+, \text{MSX}^-\right)$, where MSX^+ and MSX^- are sets of "critical" positive and negative reasons, respectively, for the preference. Ideally, an MSX will be more compact than the full RDX, while still serving as a valid certificate for the preference ordering.

More formally, let the *disadvantage* of a_1 over a_2 be $d = \sum_c I\left[\Delta_c(s,a_1,a_2) < 0\right] \cdot |\Delta_c(s,a_1,a_2)|$, (where I is the identity function) that is, the total magnitude of reasons that prefer a_2. The positive MSX component is *the smallest set of positive reasons required for a_1 to*

[1] Specifically, we must update each state-action pair infinitely often, and the learning rates $\alpha_t(s,a)$ must satisfy $\sum_t \alpha_t(s,a) = \infty$ and $\sum_t \alpha_t^2(s,a) < \infty$.

outweigh d, i.e.,

$$\text{MSX}^+ = \arg\min_{M \in 2^C} |M| \text{ s.t. } \sum_{c \in M} \Delta_c(s, a_1, a_2) > d.$$

This definition selects the smallest cardinality set of reasons whose sum overcomes the disadvantage. There may be multiple such sets M, and we break ties by preferring larger sums $\sum_{c \in M} \Delta_c(s, a_1, a_2)$. An MSX^+ can be efficiently computed by greedily adding positive reasons to the set from largest to smallest until exceeding the disadvantage. If there are ties among positive reasons, then MSX^+ may not be unique, with the number of choices being exponential in the MSX size in the worst (pathological) case. We break ties based on lexicographic ordering.

The role of MSX^- is to answer the followup question "What are the critical disadvantages of a_1 relative to a_2 that make all reasons in MSX^+ necessary?". Define the just-insufficient advantage as

$$v = \sum_{c \in \text{MSX}^+} \Delta_c(s, a_1, a_2) - \min_{c \in \text{MSX}^+} \Delta_c(s, a_1, a_2),$$

which sums all reasons in MSX^+ except the smallest. The minimal set of negative reasons with total magnitude greater than v shows that all reasons in MSX^+ are necessary, yielding our definition:

$$\text{MSX}^- = \arg\min_{M \in 2^C} |M| \text{ s.t. } \sum_{c \in M} -\Delta_c(s, a_1, a_2) > v,$$

which may not be unique. Preference is given to larger disadvantages, then lexicographic ordering.

When all reasons are needed to capture the preference, the MSX offers no compression advantage compared to the RDX. At the other extreme, when all reasons are positive in the RDX, MSX^+ and MSX^- will be empty, indicating that the agent believes that a_1 dominates a_2 in all respects.

Related Work. Prior work [8] developed a similar notion to MSX.[2] However, their formulation is stricter and will produce larger explanations. Part of the difference is that they do not present the negative reasons to the user. Instead the approach makes the pessimistic choice of assuming any reason not included in the MSX has the minimum possible value. Rather, our MSX definition aims to explain the agent in terms of the information it actually predicted for negative reasons, since that is ultimately what the preferences are based on.

5 Experimental Case Studies

Our explanation interface is fully compatible with the standard OpenAI Gym [2] environment API. The environment interface is extended to provide decomposed reward information through an auxillary channel, which can be ignored by non-decomposed RL agents. The explanation interface allows users to load one or more trained decomposition-based agents along with a saved state trajectory, usually produced by one of the agents. The user can then navigate along the trajectory, to visualize the states and action choices of each agent. At each state, the user can visualize the agents' overall and decomposed Q-values along with the RDX and MSX for a selected pair of actions.

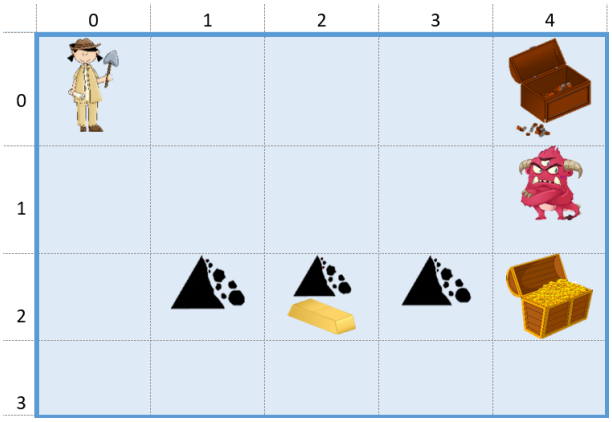

Figure 1: Picture illustrating the cliffworld environment. Cliffs are -10 points, empty treasure chest is 1, monster is -2, gold is 10, and filled treasure chest is 15.

Below we describe two case studies using the interface. The primary goal is to demonstrate the interface's utility to RL experts, which is arguably the first population to be practically impacted. In addition, to provide a quantitative evaluation of the explanations produced by the trained RL agents, the first environment is selected to be small enough to solve for the ground truth decomposed Q-values and in turn the corresponding ground truth explanations. In all cases, neural networks are trained using the Adam [9] optimizer with an initial learning rate(lr) of 0.01 and a discount factor (λ) of 0.99. ϵ-exploration is used with ϵ decaying from 0.9 to 0.1 and thereby kept constant. *The code and explanation interface will be made publicly available.*

5.1 CliffWorld

Cliffworld is a grid-world where cells can contain cliffs, monsters, gold bars, and treasure. Figure 1 shows an illustrated map of the domain.

Episodes end when encountering a **cliff**, **monster**, or **treasure**. The decomposed reward vector at each step with reward types *[cliff, gold, monster, treasure]* reflecting the current cell's contents: cliff (-10), monster (-20), gold bars (+10), upper treasure (+1), lower treasure (+15). The four actions N, E, W, S move the agent in the corresponding direction or have no effect at grid boundaries. We show results for a deterministic version of the environment, noting that the stochastic version with standard grid-world "movement noise" yielded similar results. The optimal policy is to go around and along

[2]The work did not use reward decomposition. They use an alternate metric concerning the predicted values of "templates" of states. Regardless, the general approach taken is qualitatively similar.

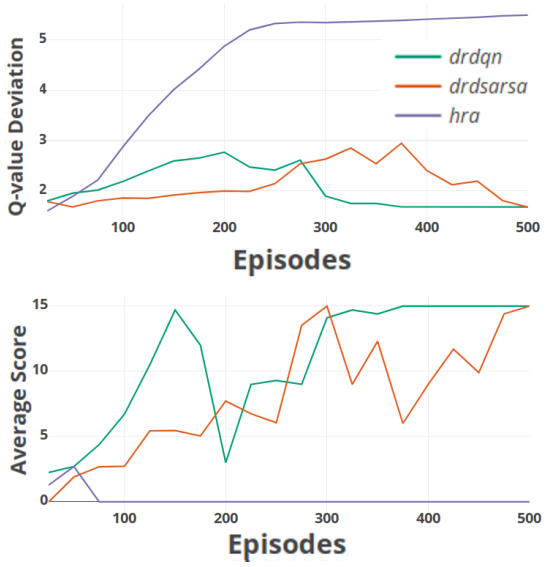

Figure 2: (Top) Deviation from ground-truth decomposed Q-values; (Bottom) Learning curves averaged over 5 runs.

a cliff range to reach the lower treasure while avoiding the monster cell.

We train a policy for each algorithm: drDQN, drDSARSA, and HRA. In all cases, we use a linear network for each reward type, each having four outputs giving the Q-value estimate for each action and the input being a one-hot encoding of the agent position in the grid.

Learning Performance. The bottom graph in Figure 2 shows the average of 5 runs of the learning algorithm with each point giving the results of 10 test games after training for the specified number of episodes. We see that drDQN and drDSARSA learn the optimal policy (verified via the ground truth). HRA fails in this domain due to its unsound training strategy as discussed further below.

Quantitative Explanation Accuracy. The top graph in Figure 2 evaluates how well the learned decomposed Q-values reflect the actual value of the current greedy policy as learning progresses (increasing episodes). To do this, we extract the policy at each point in learning and perform an exact decomposed Q-iteration (using the exact known model) to get ground-truth component Q-functions and then measure the average absolute deviation from the learned component Q-functions. drDQN and drDSARSA converge to have nearly zero deviation, while HRA exhibits a large deviation. Thus, HRA is unable to learn values that reflect it's actual policy, which means explanations can be misleading.

Explanation Insights. We were interested in understanding why HRA performs so poorly and noticed that when in cell (3,4) directly below the treasure it selects *right*, hitting the wall, and remains in that cell until the episode terminates. The correct choice is *up* to get the treasure. To understand why HRA shows this behavior we look at the component Q-values of all actions in state (3,4) for each algorithm (Figure 3). The values for drDQN and drDDSARSA are close to ground truth, while we see HRA has much different estimates. HRA predicts similarly for the "treasure" reward, yet believes it will gain an additional "gold" reward for taking any of the three non-terminal actions. This is puzzling at first, but HRA's Q-function for gold learns that even after going right it can then go get the gold in the cliffs as well as also get the treasure, but if it goes up the episode is over and it will get only the treasure. This is one example that exhibits the inconsistency of HRA, which is only possible to fully observe by viewing the decomposition.

5.2 Lunar Lander

Lunar Lander involves controlling a rocket during a ground landing. The actions are to fire any of its three engines: main, left, right engine, or no-op. The standard implementation has the following natural reward decomposition: 1) crashing penalty, 2) safe landing bonus ("live"), 3) main-engine fuel cost, 4) side-engines fuel cost, and 5) shaping reward. Episodes end with either a safe landing (+100 reward) or crashing (-100 reward). We slightly modified the simulator to make the components explicit and also further decomposed the shaping reward into its own natural components: 5.1) destable angle of the rocket, 5.2) contact between legs and ground, 5.3) reward for being closer to landing pad, 5.4) penalty for high velocity.

The neural networks had 1 hidden layer of 64 ReLU units and an output layer with one linear unit per action. The input to the network is a state describing lander's current position, velocity, orientation, angular velocity, and leg-ground contact information. During training, drDQN and drDSARSA were able to achieve a good policy of consistently landing safely, whereas HRA's policy continuously degraded. The graphs are omitted due to being qualitatively similar to Cliffworld.

Shaping Rewards Dominate. Using the interface we were surprised to find that the the policies drDQN and drDSARSA are almost entirely governed by the shaping reward components. A typical example is in Figure 4 (right), corresponding to a state near the landing pad, where the agent should be mostly concerned with the crash prevention reward. However, the MSX is governed by the "velocity" and "landing-pad distance" shaping rewards. The true environmental reward hardly has an influence on the decisions. This was generally the case at other decision points. As an RL practitioner, it is important to realize that the shaping reward is dominating the environment reward, since ultimately we would like to optimize environment reward. *To the best of our knowledge, our proposed explanation approach is the only way to make such observations.*

Optimistic HRA. HRA doesn't learn to land and prefers to crash itself. It does that either by rising-up and moving out of the frame or by toppling itself to fall on the ground. Why does it do this? Figure 4 (left)

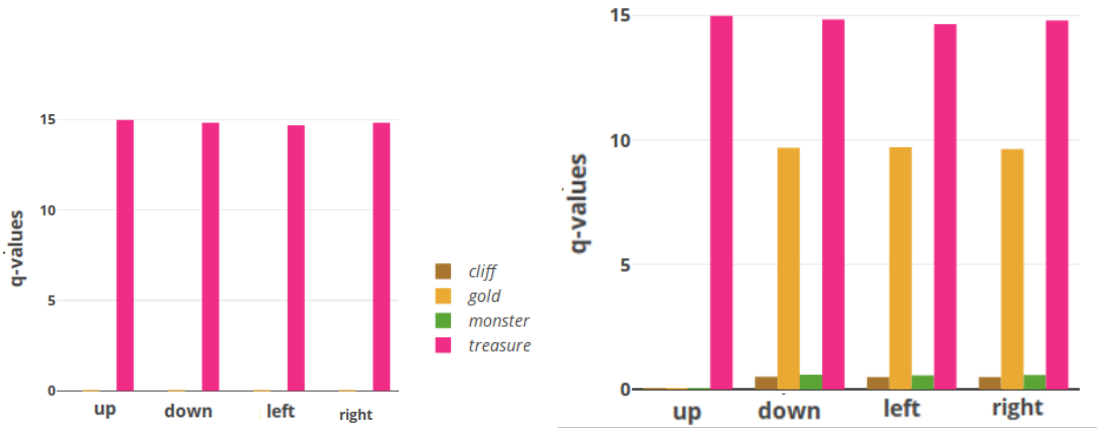

Figure 3: Reward decompositions for DQN (left) and HRA (right) at cell (3,4) in Cliffworld. SARSA is omitted but is qualitatively identical to DQN. HRA predicts an extra "gold" reward for actions which do not lead to a terminal state.

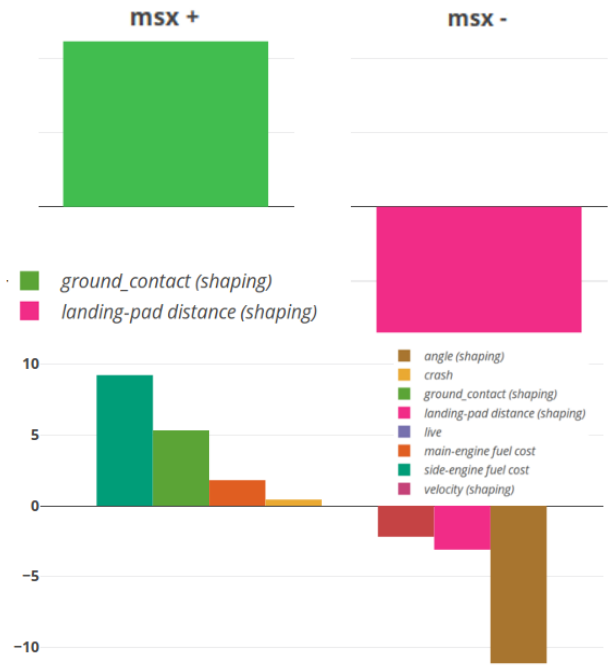

Figure 4: (top) MSX (fire-main-engine vs. noop) for drDQN in Lunar Lander near landing site. The shaping rewards dominate decisions. (bottom) RDX (noop vs. fire-main-engine) for HRA in Lunar Lander before a crash. The RDX shows that noop is preferred to avoid penalties such as fuel cost.

gives insight into why HRA exhibits this behavior. In this case, HRA prefers to do a noop rather than fire an engine, despite that leading to a crash. The RDX shows that it prefers noop to avoid penalties for fuel-cost, high-velocity, and de-stabilization of the lander. In other words, HRA decides to end its life quickly to avoid the penalties. This again is a result of HRA's unsound independent optimization of each component.

Error Boosting Optimizer. Looking at the decomposed Q-values for HRA we also see surprisingly huge positive Q-value estimates for reward types that give only negative rewards (for example the velocity penalty). We would expect the predicted Q-values for these penalty reward types to always be negative. We investigated this anomaly and found that during gradient optimization, Adam, via its adaptive learning rate, can lead to small positive values early on in learning. Since these values are used in the target values during learning updates, the error can get amplified. This illustrates how the use of decomposition allowed for the identification of a relatively serious error. *It is unclear how this component-level sign inconsistency would be noticed without learning decomposed Q-values.*

6 Summary

In many environments, the reward function has a natural decomposition into meaningful components, but this structure is typically ignored in RL. This paper introduced an approach for leveraging such decompositions to help explain the action preferences of RL agents. We gave the first proof of a convergent off-policy RL algorithm for this setting, which overcomes deficiencies in prior approaches for learning from decomposed rewards. We provided case studies in two environments to illustrate the potential of our visual explanations in the hands of an RL practitioner. This allowed for spotting certain "bugs" in the agent's action values and even help identify a more fundamental issue related to the interaction of the gradient optimizer and the RL loop. In important concurrent work, we are also conducting a large scale study involving non-expert end-users to evaluate the utility for that very different, but also important, population.

Acknowledgments

This work is partially supported by the National Science Foundation under the grant CCF-1717300 and by DARPA under the grant N66001-17-2-4030.

References

[1] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[3] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[4] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pages 2112–2121, 2018.

[5] Stuart J. Russell and Andrew Zimdars. Q-decomposition for reinforcement learning agents. volume 2, pages 656–663, 01 2003.

[6] Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. Convergence of stochastic iterative dynamic programming algorithms. In *Advances in neural information processing systems*, pages 703–710, 1994.

[7] Jonas Karlsson. Task decomposition in reinforcement learning. In *Proceedings of the AAAI Spring Symposium on Goal-Driven Learning, Stanford, CA*, 1994.

[8] Omar Khan, Pascal Poupart, and James Black. Minimal sufficient explanations for factored markov decision processes. In *International Conference on Automated Planning and Scheduling*, 2009.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Jelle R Kok and Nikos Vlassis. Sparse cooperative q-learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 61. ACM, 2004.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[12] OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018.

[13] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[14] Tianmin Shu, Caiming Xiong, and Richard Socher. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. In *International Conference on Learning Representations*, 2018.

[15] Chen K Tham and Richard W Prager. A modular q-learning architecture for manipulator task decomposition. In *Machine Learning Proceedings 1994*, pages 309–317. Elsevier, 1994.

[16] Harm van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid reward architecture for reinforcement learning. *CoRR*, abs/1706.04208, 2017.

[17] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.