Reinforcement Learning via Reasoning from Demonstration

Lisa Torrey
St. Lawrence University

ABSTRACT

Demonstration is an appealing way for humans to provide assistance to reinforcement-learning agents. Most approaches in this area view demonstrations primarily as sources of behavioral bias. But in sparse-reward tasks, humans seem to treat demonstrations more as sources of causal knowledge. This paper proposes a framework for agents that benefit from demonstration in this human-inspired way. In this framework, agents develop causal models through observation, and reason from this knowledge to decompose tasks for effective reinforcement learning. Experimental results show that a basic implementation of Reasoning from Demonstration (RfD) is effective in a range of sparse-reward tasks.

1 INTRODUCTION

With reinforcement learning (RL), agents can train themselves autonomously to complete tasks. RL algorithms can allow agents to develop successful behaviors based only on environmental feedback. However, tasks in large environments with infrequent feedback, known as sparse-reward tasks, have always been difficult for RL agents to learn.

Demonstration is a practical and human-motivated approach to coping with challenging RL tasks. Learning from demonstration (LfD) can either replace RL or augment it. Systems that combine LfD and RL typically do so by encouraging agents to behave more like demonstrators as they learn. This usually improves their early performance, which would otherwise be essentially random.

It can be effective to use demonstrations as models of behavior. This paper asks whether it could also be effective to use demonstrations as sources of knowledge. In sparse-reward tasks, knowledge seems particularly valuable.

Consider, in the classic Atari game Montezuma's Revenge (see Figure 1), the task of exiting the first room without losing a life. It takes a sequence of several hundred carefully chosen actions to complete this task. The deep Q-learning system that initially mastered many Atari games [25] did not learn to complete it. Yet the task is simple for a human to understand: the player must navigate to a key, and then to a door, without falling off any platforms, and without colliding with a rolling skull.

This paper proposes a framework in which agents acquire this kind of high-level knowledge by observing a demonstration, and then to use that knowledge to learn the task more effectively.

Recent studies show that rendering video games with textured cells rather than recognizable objects hinders human learning significantly [13]. One plausible explanation is that object perception facilitates causal model-building, which cognitive scientists recognize as a critical component of human learning [22]. This paper asks how LfD agents might incorporate causal reasoning, and thereby benefit from demonstration more like humans seem to do in tasks like Montezuma's Revenge.

2 BACKGROUND

This section presents a short introduction to RL and surveys techniques for making it more effective in challenging tasks. It focuses on single agents learning single tasks, and even within this limited context it is far from exhaustive, but it provides the necessary context for the proposals that follow.

2.1 Reinforcement learning

An RL agent develops a policy for choosing actions based on the state of its environment. It receives a numeric reward from the environment as feedback for each step. Balancing between exploiting its developing policy and exploring alternatives, the agent gradually improves its policy and becomes capable of earning higher cumulative rewards [31].

Q-learning [37] is one of the simpler RL algorithms. It uses a function $Q(s, a)$ to estimate the cumulative reward an agent can expect to earn after taking action a from state s. Given an accurate Q-function, the optimal action in state s is the one that maximizes $Q(s, a)$. Agents can begin with all $Q(s, a) \approx 0$ and then adjust the Q-values incrementally. After taking action a in state s, receiving reward r, and transitioning to state s', the Q-learning update is:

$$Q(s, a) \longleftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a')) \qquad (1)$$

The α parameter is a learning rate, which controls the update size. The γ parameter is a discount factor, which controls the agent's patience for delayed rewards. A simple exploration strategy for Q-learning is ϵ-greedy action selection: the agent usually takes the action with the highest Q-value, but with probability ϵ it chooses a random action instead.

2.2 Sparse rewards

RL excels in environments with frequent informative rewards, which allow agents to improve their policies steadily. As non-zero rewards become less frequent, it takes longer to discover them, and it takes more repetition to determine which actions are responsible for them. Sparse rewards cause combinatorial explosions in the training time required to develop competent agents.

Taxi [9] is a small example of a sparse-reward task. In this task, a taxi moves through a 5x5 grid world (see Figure 1), picking up a passenger at one stop and dropping it off at another, with both stops randomly selected from the four corners of the grid. The agent does receive a small negative reward (-1) after each action, to discourage loitering, but it only receives a positive reward (+20) once it completes the task. Each episode ends after 200 actions, whether or not the task is complete.

Courier (see Figure 1) is a scaled-up version of Taxi, with similar dynamics but significantly increased size and sparsity. In this task, a courier moves through a 35x35 grid world, collecting four randomly-placed packages and delivering them to a central platform. There is no time limit on episodes, but there are 20 randomly-placed moving

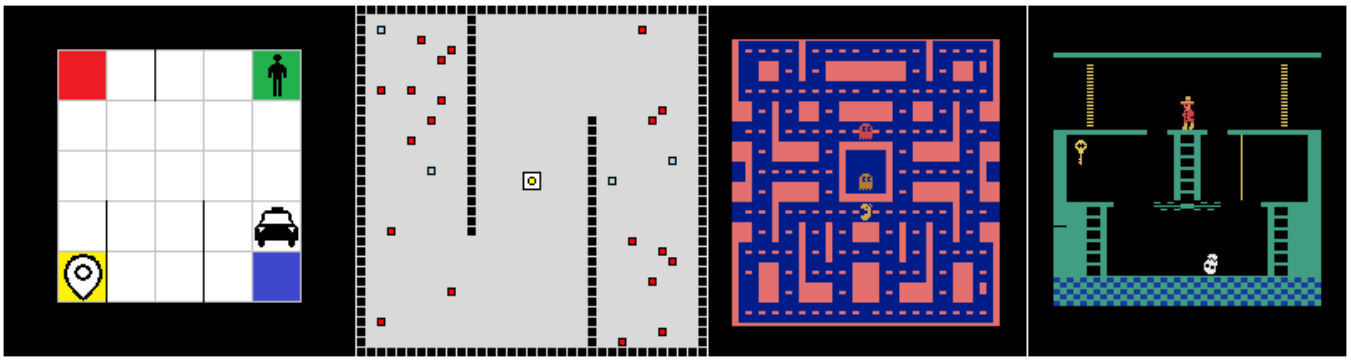

Figure 1: Renderings of four virtual environments: Taxi, Courier, Ms. Pacman, and Montezuma's Revenge.

vehicles that the agent must dodge. The only feedback occurs at the end of the task: *success* when the courier delivers the last package, or *failure* if it gets run over.

More examples of sparse-reward tasks can be found in the Arcade Learning Environment [24], which emulates classic Atari games with large and unstructured state spaces (160x210 pixel arrays). Figure 1 illustrates two of these games. Montezuma's Revenge has one of the most naturally sparse reward systems; in the first room, the only rewards are +100 for acquiring the key and +300 for exiting the room. Ms. Pacman has frequent small rewards for eating pellets, but the largest rewards can only be achieved by generating and eating multiple edible ghosts.

2.3 Scaling techniques

Standard Q-learning, which assigns a value to every state-action pair, is only practical in small environments like Taxi. Larger environments, like Courier and the Atari games, have too many states. This problem can be addressed by decomposing states into feature vectors and replacing Q-tables with approximators, which require less memory and allow for generalization across states. Q-learning adapts well to linear and neural approximators using updates derived from gradient descent [31].

There is a long history in RL of engineering features that alter the state space, either to abstract away irrelevant information or to compose useful information. For example, with sophisticated features, linear Q-learning is sufficient to achieve high scores in the first level of Ms. Pacman [34]. However, in the era of deep learning, the field has moved away from human feature engineering in favor of implicit feature construction via convolutional neural networks. Deep Q-networks can reach human-level play in many Atari games directly from their raw pixel states [25]. But games like Montezuma's Revenge remain difficult for deep Q-learning because of their sparse rewards.

One way to cope with sparse rewards is to shape them. Reward shaping is a technique for providing extra rewards to encourage progress towards task completion. Shaped rewards must be designed appropriately in order to be effective [26]. For example, on an open grid, it would be effective to reward a taxi by the amount that it reduces its Manhattan distance from its destination, but this intuitive shaping strategy would be counter-productive on the actual Taxi grid, which contains walls that restrict movement. A recent and more abstract form of reward shaping, called intrinsic motivation, encourages exploration by rewarding agents for reaching new areas of the state space; this technique has some success in Montezuma's Revenge [7].

Another way to cope with sparsity is to decompose tasks into subtasks and apply hierarchical learning. In the options framework [32], subtask policies are treated like extended actions, which may sometimes be selected instead of primitive actions. Other frameworks, like Feudal RL [8] and MAXQ [9], more explicitly formulate tasks as hierarchies of subtasks.

Like features and rewards, task decompositions have often been engineered. For example, one approach uses human domain knowledge to define a separate learner for each reward source, and chooses actions by aggregating their Q-values [35]. Given some human engineering of states and rewards as well, this system is capable of outplaying humans in Ms. Pacman. Other approaches train low-level learners to control transitions in a higher-level plan, which is derived from abstract states [27] or symbolic rules [23] defined by a human. Both of these techniques are effective in Montezuma's Revenge.

There are also approaches in which agents learn to decompose tasks themselves. For example, one method for learning options has been applied to Ms. Pacman [1], and one for learning feudal hierarchies has been applied to Montezuma's Revenge [36]. These systems can outperform their non-hierchical baselines, but their scores do not yet compete with the human-guided approaches.

2.4 Objects

Many tasks are defined around objects. Important objects tend to be visually displayed for human players, as in Figure 1, but they are not always explicitly described to RL agents. For example, Taxi traditionally decomposes states into four features: the row and column numbers of the taxi location, and the stop numbers of the passenger and destination. These values uniquely specify each state, but leave much of the visual information implicit. At the other extreme, the pixel states of Atari games provide complete visual information, but in a completely unstructured form.

Some RL algorithms are designed specifically for object-oriented states. Relational RL [14] trains a first-order logical Q-function approximator, using states that are sets of first-order predicates, grounded with objects in the environment. Object-oriented RL

[12] uses a similar state representation, though not a first-order Q-function, to speed up learning in Taxi.

Object-oriented states can provide natural opportunities for task decomposition. For example, Object-Focused Q-learning [5] trains a policy for interacting with each type of object, and chooses actions by aggregating their Q-values. Hierarchical Deep Q-learning [21] trains one network to select an object, and another to navigate towards a selected object; this is another approach that is effective in Montezuma's Revenge.

Of all the forms of state engineering involved in RL, object-oriented states are arguably the most intuitive, since humans perceive objects with so little apparent effort. Furthermore, object tracking is an active area of research in the field of computer vision, and there have been steps towards applying it to object-focused Q-learning [17].

2.5 Demonstration

LfD originated in the field of robotics, where demonstrations are a natural form of assistance for humans to provide, and it can be particularly slow or costly to allow agents to explore autonomously. In this context, LfD may be a form of supervised learning, with demonstrations as the source of training data. For example, an agent can approximate the demonstrator's policy, and use the approximation as its own policy [29].

In environments where exploration is permissible, LfD may also involve autonomous practice. RL provides opportunities to acquire better and more complete policies than can be acquired from demonstration alone. For example, an agent can use an approximate demonstrator policy as a starting point, and then train it further with RL [30]. Alternatively, the agent can develop its own policy from scratch, but use an approximate demonstrator policy to choose some actions in the early stages of training [33]. Failed demonstrations as well as successful ones can be leveraged to form useful training biases [18].

Demonstrations can also bias the policies of RL agents less directly. For example, one approach shapes rewards for agent actions based on how similar they are to demonstrated actions [2]. Another system includes demonstrated transitions in the batches that it collects to update a deep Q-network [19]. In one curriculum-based approach, the agent learns autonomously, but using demonstration states as starting points [28]; this is another effective strategy for Montezuma's Revenge.

Some LfD approaches use demonstration to facilitate decomposition. For example, one method achieves state abstraction by removing features that appear to be irrelevant to demonstrator actions [6]. A subsequent method goes on to define subtasks based on differences in their state abstractions [4]. Several approaches construct option policies based on segments of demonstrations [16, 20, 39]. In the context of robotics, demonstrations have been used to teach object affordances, which roughly correspond to macro-actions that robots can perform upon objects [3].

Decomposing tasks via demonstration is arguably an appealing middle ground between designing subtasks by hand and learning them from scratch. Demonstration often requires less expertise than engineering. And even humans are rarely expected to learn complex tasks without seeing them demonstrated first.

3 REASONING FROM DEMONSTRATION

What would a human learn from a demonstration of Montezuma's Revenge? For an anecdote, I asked a colleague with no history of gaming to watch and describe a YouTube video of the first room. He reported that "the cowboy had to go get something and bring it up there." Compared to what most LfD agents acquire from demonstration, what my colleague retained seems at once much less and much more. He would likely recall no specific states or actions, but he could explain how to succeed in this task.

This paper asks how an LfD agent could acquire this kind of high-level knowledge from a demonstration, and how it could use that knowledge as it trains. Since no previous LfD systems appear to be based on these questions, this is a fundamental difference between the proposed approach and the existing ones. Let us call this approach Reasoning from Demonstration (RfD).

The goal of an RfD agent is to benefit from demonstration more like humans seem to do. The core of the approach is causal model-building based on objects and their interactions. Given a demonstration, an RfD agent generates a set of cause-effect hypotheses. Causes are object interactions, and effects are other observable events, such as object appearances and environment feedback. Using these hypotheses, an RfD agent can identify desirable and undesirable object interactions. This provides a basis for task decomposition.

The rest of this section describes an RfD agent that illustrates one possible implementation of the broader concept. After motivating the agent's design, the section introduces formal notation and procedures. To facilitate reproduction and future work, all of the code for this agent is available on GitHub[1], along with all of the data generated in the experiments further below.

3.1 Objectives and anti-objectives

Let an *objective* be an object interaction that contributes towards success in the task. For example, in Montezuma's Revenge, the first objective should be to bring the main character, Panama Joe, into contact with the key, causing the door to become openable. The second objective should be to bring Joe into contact with the openable door, causing the success feedback. After observing these events in a demonstration, any RfD agent should form hypotheses about causes and effects, and these hypotheses should allow it to reason about objectives.

Humans seem to instinctively focus our attention and monitor our progress in object-oriented ways. For example, if we are trying to bring Joe to the key, we will focus primarily on those two objects, and we will be aware that we are making progress as they get closer together. The RfD agent presented in this paper therefore performs object-oriented state abstraction and distance-based reward shaping as it pursues objectives.

Naive distance-based shaping would encourage straight-line movement, which is not always correct. For example, Joe's only viable path to the key nearly circles the entire room. Progress towards objectives needs to be judged along paths that respect the environmental terrain. The RfD agent presented in this paper therefore develops a map of the region connectivity of its environment, so that it can shape rewards effectively.

[1] https://github.com/lisatorrey/reasoning-from-demonstration

Let an *anti-objective* be an object interaction that causes failure. For example, the task in Montezuma's Revenge fails if Joe comes into contact with the skull. Since a successful demonstration would not convey this knowledge, it must be discovered during autonomous practice. Any RfD agent should therefore continue to acquire cause-effect hypotheses as it trains.

Anti-objectives need to be avoided during the pursuit of objectives. However, there is neurological evidence that human brains process risks separately from rewards [38]. The RfD agent presented in this paper therefore develops separate policies with respect to objectives and anti-objectives, and consults both when choosing actions.

3.2 Objects and events

Let an *environment* consist of a state space, an action space, and an unknown probability distribution over state transitions. Let a *task* be defined over an environment by choosing four subsets of the state space: the state sets where task attempts begin, end, receive `SUCCESS` feedback, and receive `FAILURE` feedback. In this task formulation, there are no intermediate rewards.

Any RfD agent needs to be able to perceive objects and events in its environment, but agents may differ in how they represent these elements. Here is the representation used by the RfD agent presented in this paper.

Let `objects(`s`)` be the set of objects present in state s. Given an object, let `type(object)` be a descriptor that similar objects would share. Let `location(`s`, object)` and `velocity(`s`, object)` describe the perceived position and momentum of the object in state s. Let `region(`s`, object)` indicate which region of the environment the object occupies in state s.

Let `events(`s`,` s'`)` be the set of object interactions observed at the transition from state s to s'. Given an event, let `type(event)` be a descriptor that similar events would share. Let `actor(event)` be the object responsible for the event, and let `subject(event)`, if it exists, be the object acted upon.

Given an event, let `template(event)` be composed of three types: `type(event)`, `type(actor(event))`, and `type(subject(event))`. Let a template be called *instantiable* in states that contain appropriately typed objects to fill the actor and subject roles. Let instantiated templates be called *possible events*. Let `instances(`s`, templates)` be the set of possible events generated by instantiating one or more templates in state s.

Given a possible event, let `distance(`s`, event)` be the distance between its objects in state s. Let `focus(`s`, event)` be an abstract state representation for the possible event. In this paper, abstract states consist of the velocities of the objects and the location of the actor (relative to the subject, if there is one) in state s.

3.3 Training procedures

The main procedure for the RfD agent presented in this paper is Algorithm 1, which augments the standard RL loop for episodic training. The agent develops three kinds of knowledge: a theory, a map, and a set of policies. It begins to develop some of these components based on a demonstrated state sequence, and improves them all as it practices the task autonomously.

Algorithm 1 Outlines an RfD agent.

```
procedure RFD
    for each demonstrated transition (s, s′) do
        UPDATE-MAP(s, s′)
        UPDATE-THEORY(s, s′)
    while more training is desirable do
        s ⟵ random choice from initial task states
        while s is non-terminal and attempt time < τ do
            anti-objectives ⟵ instances(s, causes(FAILURE))
            objectives ⟵ instances(s, CONTRIBUTORS(SUCCESS))
            objective ⟵ CHOOSE-OBJECTIVE(objectives)
            if objective is multi-regional then
                objective ⟵ FIRST-CHECKPOINT(objective)
            a ⟵ CHOOSE-ACTION(s, objective, anti-objectives)
            s′ ⟵ result of taking action a
            UPDATE-MAP(s, s′)
            UPDATE-THEORY(s, s′)
            UPDATE-POLICIES(s, a, s′, objective, anti-objectives)
            s ⟵ s′
```

Algorithm 2 Makes a theory consistent with a transition (s, s').

```
procedure UPDATE-THEORY(s, s′)
    for each event in events(s, s′) do
        if template(event) has not been seen before then
            for each object in objects(s′) − objects(s) do
                add template(event) ⟶ type(object) to the agent's theory
            if SUCCESS occurred in s′ then
                add template(event) ⟶ SUCCESS to the agent's theory
            if FAILURE occurred in s′ then
                add template(event) ⟶ FAILURE to the agent's theory
    for each cause ⟶ effect in the agent's theory do
        if events(s, s′) contains event s.t. template(event) = cause then
            if effect did not occur in s′ then
                remove cause ⟶ effect from the agent's theory
```

Algorithm 3 Generates objectives contributing to an effect E.

```
procedure CONTRIBUTORS(s, E)
    templates ⟵ ∅
    for each C in causes(E) do
        if C is instantiable in s then
            add C to templates
        else
            for each object-type required to make C instantiable in s do
                merge CONTRIBUTORS(s, object-type) into templates
    return templates
```

A *theory* is a set of cause-effect hypotheses. The agent constructs a theory through repeated application of Algorithm 2, which reflects a few simple assumptions about causality. It assumes object interactions are potential causes, while object appearances and environment feedback are potential effects. Furthermore, it assumes causes and effects coincide in time; if it observes potential cause C and potential effect E in the same state, it generates the hypothesis $C \longrightarrow E$. Finally, it assumes the rule of logical implication; if it observes C but not E, it rejects the hypothesis $C \longrightarrow E$.

The agent uses its theory to identify objectives and anti-objectives. Anti-objectives are possible events to be avoided, since they may cause failure. Objectives are possible events to be pursued, because they contribute along some causal path towards success, as identified in Algorithm 3. In this algorithm, `causes(`E`)` indicates the set of all C such that $C \longrightarrow E$ is in the agent's theory.

A *map* is a graph of the regions of the environment and their connectivity. The agent constructs this map through repeated application of Algorithm 4, which keeps track of where objects move

Algorithm 4 Adds to a map based on a state transition (s, s').

```
procedure UPDATE-MAP(s, s′)
    for each object in objects(s) ∩ objects(s′) do
        R ⟵ region(s, object)
        R′ ⟵ region(s′, object)
        if R ≠ R′ and no transition R ⟶ R′ exists in the agent's map then
            add transition R ⟶ R′ to the agent's map at location(s′, object)
```

across regions. To keep the map small, the agent stores just one transition for each pair of regions.

The agent uses its map to break down multi-region objectives, which involve objects in different regions. It applies Dijkstra's algorithm [11] to find the shortest path from actor to subject, using region-transitions as intermediate points. The length of a path is the sum of the lengths of its segments. The FIRST-CHECKPOINT procedure constructs an intermediate objective of moving the actor to the first region-transition along this path. The agent also uses its map whenever there are multiple objectives to choose from; its CHOOSE-OBJECTIVE procedure selects an objective with minimal path length, breaking ties randomly. (The FIRST-CHECKPOINT and CHOOSE-OBJECTIVE subprocedures are the only ones for which pseudocode is omitted due to space constraints.)

A *policy* evaluates actions with respect to a possible event. The agent represents all of its policies with simple Q-functions, and associates one Q-function with each event template. Whenever it discovers a new event template, it creates a new zero-valued Q-function. As it trains, it generates its own rewards to update the appropriate Q-functions, as shown in Algorithm 5. Objective and checkpoint rewards are based on progress, and all policies have a bonus ω for completion.

The agent chooses actions by consulting the policies of its current objective and anti-objectives. It uses the corresponding Q-functions to compute both a reward estimate and a cumulative risk estimate for each action, as shown in Algorithm 6. When it decides to explore, it randomly chooses a minimal-risk action. Otherwise, it chooses an action that maximizes the gap between reward and risk, using a weight β to control risk tolerance.

Exploration rates (ϵ) and risk tolerances (β) are specific to the current objective. New Q-functions for objectives begin with $\epsilon_Q = \epsilon_{max}$. Whenever an objective is completed, its ϵ decays, with a floor of ϵ_{min}. The agent therefore explores less the more an objective succeeds. New Q-functions for objectives also begin with $\beta_Q = \beta_{max}$. Whenever an objective is exploited, its β decays, but whenever it is completed, its β resets to β_{max}. The agent therefore tolerates more risk the longer it pursues an objective without completing it.

All of the experiments in this paper use the following settings for the Greek-letter parameters: $\alpha = 0.1$, $\gamma = 0.9$, $\omega = 100$, $\tau = 10000$, $\epsilon_{max} = 0.1$, $\epsilon_{min} = 0.01$, $\lambda_\epsilon = 0.99$, $\beta_{max} = 100$, $\lambda_\beta = 0.99$.

4 BENCHMARK

Taxi makes a good benchmark for comparing RfD with LfD. It is small enough to apply any approach, but sparse enough to reveal differences between approaches. There are two main categories of LfD that should be considered: approaches that use demonstration to influence a single policy, and approaches that use demonstration to decompose the task. This section identifies one approach in each

Algorithm 5 Adjusts policies after a transition (s, a, s').

```
procedure UPDATE-POLICIES(s, a, s′, objective, anti-objectives)
    Q ⟵ the Q-function associated with template(objective)
    δ ⟵ distance(s, objective) − distance(s′, objective)
    s ⟵ focus(s, objective)
    s′ ⟵ focus(s′, objective)
    if objective is a checkpoint then
        if actor(objective) has entered the checkpoint region then
            update Q according to Equation 1 with s, a, s′, and r = δ + ω
            ε_Q ⟵ max(ε_min, λ_ε ε_Q)
            β_Q ⟵ β_max
        else if actor(objective) has entered a different region then
            update Q according to Equation 1 with s, a, s′, and r = δ − ω
        else
            update Q according to Equation 1 with s, a, s′, and r = δ
    else
        if objective ∈ events(s, s′) then
            update Q according to Equation 1 with s, a, s′, and r = δ + ω
            ε_Q ⟵ max(ε_min, λ_ε ε_Q)
            β_Q ⟵ β_max
        else if objective remains possible in s′ then
            update Q according to Equation 1 with s, a, s′, and r = δ
    for each anti-objective do
        Q ⟵ the Q-function associated with template(anti-objective)
        s ⟵ focus(s, anti-objective)
        s′ ⟵ focus(s′, anti-objective)
        if anti-objective ∈ events(s, s′) then
            update Q according to Equation 1 with s, a, s′, and r = −ω
        else if anti-objective remains possible in s′ then
            update Q according to Equation 1 with s, a, s′, and r = 0
```

Algorithm 6 Chooses an action in state s.

```
procedure CHOOSE-ACTION(s, objective, anti-objectives)
    for each a in the action space do
        risk[a] ⟵ 0
    for each anti-objective do
        Q ⟵ the Q-function associated with template(anti-objective)
        s ⟵ focus(s, anti-objective)
        risk[a] ⟵ risk[a] − Q(s, a)
    Q ⟵ the Q-function associated with template(objective)
    s ⟵ focus(s, objective)
    for each a in the action space do
        reward[a] ⟵ Q(s, a)
    if random(0,1) < ε_Q then
        safest ⟵ set of a minimizing risk[a]
        return random choice from safest
    else
        best ⟵ set of a maximizing reward[a] − β_Q risk[a]
        β_Q ⟵ λ_β β_Q
        return random choice from best
```

category that seems best suited to the Taxi problem, and shows how these two LfD approaches compare with RfD.

It is easy to generate good demonstrations for Taxi. Training a standard Q-learner to convergence takes about 100,000 actions, and then it makes a reliable demonstrator. However, a Taxi demonstration provides very limited information. There are 500 unique states in the environment, and good demonstrations encounter only 5 to 20 of them. Furthermore, the traditional state representation provides little opportunity to generalize beyond a demonstration. States that match in some or even most of their feature values still often have different optimal actions.

In this situation, a single-policy LfD agent should benefit most by favoring imitation over generalization. The imitation agents in this experiment therefore use standard Q-learning, but always take demonstrated actions in demonstrated states, unless they are exploring. Figure 2 confirms that demonstrations are useful in Taxi:

the imitation agents learn faster than the standard Q-learner, and the effect increases with the number of demonstrations.

Taxi does provide opportunities for task decomposition, as well as further state decomposition within subtasks. Among the existing approaches, Automatic Decomposition and Abstraction (ADA) seems best suited to take advantage of both opportunities [4].

ADA partitions a state space into subtasks along numeric feature boundaries, and removes irrelevant features in subtasks based on their mutual information with demonstrator actions. In Taxi, the `passenger` feature has values in $\{0, 1, 2, 3, 4\}$, where 4 means the passenger is inside the taxi, and the other values represent the stops. Thus it is appropriate to use the boundary `passenger` < 4 to divide Taxi into two subtasks. One side is the pickup subtask, in which the `destination` feature should be ignored. The other side is the dropoff subtask, in which the `passenger` feature can be ignored.

In 100 trials of up to 32 demonstrations generated one at a time by a trained Q-learner, ADA produced the correct decomposition 72 times, after a minimum of 3 demonstrations and an average of 12. To establish an upper bound on the potential of ADA, the decomposition agents in this experiment all use the correct decomposition, and they train the two subtasks via imitation. Figure 2 confirms that decomposition is useful in Taxi: given the same number of demonstrations, decomposition agents quickly outperform imitation agents.

The RfD agents each receive just one demonstration, which I performed from the initial state shown in Figure 1. Instead of the usual numeric rewards of the Taxi environment, the RfD agents receive only `SUCCESS` or `FAILURE` feedback at the end of each attempt. The regions of the environment are the five rectangles suggested by the walls. Each object has a `(row, column)` location and a velocity of 0. The object types and event templates are evident from these hypotheses, which the RfD agents generate for Taxi:

picks(Taxi, Passenger) ⟶ Taxi+Passenger
picks(Taxi, Passenger) ⟶ Stop
drops(Taxi+Passenger, Stop) ⟶ Taxi
drops(Taxi+Passenger, Stop) ⟶ Passenger
drops(Taxi+Passenger, Destination) ⟶ SUCCESS
drops(Taxi+Passenger, Destination) ⟶ Stop

Differences in the forms and amounts of assistance that the RfD and decomposition agents receive complicate the comparison of their learning curves. That said, the differences in their learning curves are dramatic. The RfD agents converge in about 2% of the time that the decomposition agents take, regardless of the number of demonstrations. Figure 3 shows the RfD learning curves on an appropriate scale.

Although they represent the information differently, the RfD and decomposition agents are both solving the same subtasks in the same state space. The difference in their performance is mainly attributable to the reward shaping that the RfD agents perform. Of course, it is no surprise that reward shaping speeds up learning. But agents need causal and spatial knowledge in order to perform reward shaping themselves, without human engineering.

Overall, the comparison of ADA and RfD suggests that decomposition based on causal reasoning, rather than statistical reasoning, has the potential to facilitate significantly faster learning after significantly less demonstration.

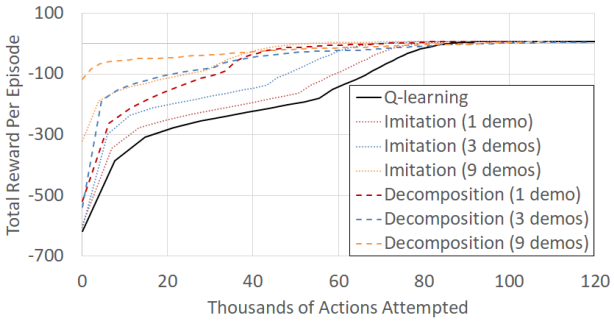

Figure 2: Learning curves for non-RfD agents in Taxi. Curves are smoothed over 400 episodes and averaged over 10 different agents.

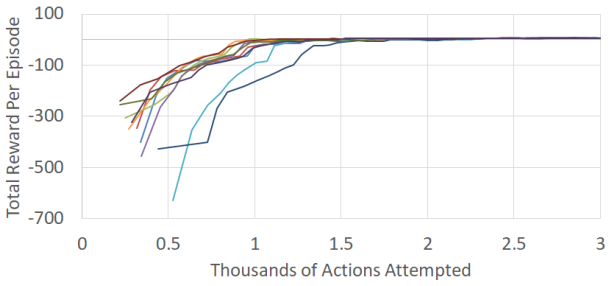

Figure 3: Learning curves for ten RfD agents in Taxi. Curves are 300 attempts long and smoothed over a 30-attempt window.

5 CHALLENGES

This section shows how RfD scales to larger and sparser environments: Courier, Ms. Pacman, and Montezuma's Revenge.

5.1 Courier

With four packages and 20 vehicles on a 35x35 grid, just the *initial* states of the Courier task number more than 10^{35}. My single demonstration begins at the state shown in Figure 1, collects the two packages on one side, collects the two packages on the other side, and delivers them all at once.

The regions of the environment are the three rectangles suggested by the walls. Each object has a `(row, column)` location. The object types and event templates are evident from these initial hypotheses that the RfD agents generate for Courier:

arrives(Courier, Package) ⟶ Courier+1
arrives(Courier+1, Package) ⟶ Courier+2
arrives(Courier+2, Package) ⟶ Courier+3
arrives(Courier+3, Package) ⟶ Courier+4
arrives(Courier+4, Platform) ⟶ SUCCESS
arrives(Courier+4, Platform) ⟶ Platform+4
arrives(Courier+4, Platform) ⟶ Courier

Agents add additional hypotheses as they train. Most importantly, they discover that collides(Courier+k, Vehicle) causes `FAILURE` for any k. They also tend to discover the possibility of delivering fewer than four packages at once. These discoveries lead some agents to develop alternate strategies.

The ADA approach is not practically applicable in Courier due to the dimensionality of the state space. And because Courier is essentially designed to need object-oriented decomposition, no other LfD approach seems more applicable. However, Figure 4 shows that training RfD agents to convergence in Courier takes less than 100,000 actions. These results indicate that combinatorially huge environment sizes and arbitrarily long subtask sequences are surmountable via RfD. Furthermore, they show that RfD agents can develop logical solutions that were not demonstrated.

5.2 Ms. Pacman

In Ms. Pacman, agents can accumulate rewards through a sequence of levels just by eating densely placed pellets. Regular pellets are worth 10 points, while the rarer power-pellets are worth 50 points. However, the highest rewards are only available for a short period after consuming a power-pellet, when the ghosts become temporarily edible. Catching one turns it into a pair of eyes and yields 200 points, and this reward doubles for each additional catch during the same period.

Let us define a task in Ms. Pacman that focuses on catching edible ghosts in the first level. It provides `SUCCESS` feedback when Ms. Pacman catches an edible ghost and `FAILURE` feedback when she collides with a regular ghost. The task ends upon failure, but success is achievable up to 16 times per attempt, because there are four power-pellets and four ghosts.

This game always begins in the same state, and the Atari emulator is, unfortunately, inherently deterministic; the same sequence of player actions always produces the same sequence of states. A common way to introduce some stochasticity is to delay the start of the game, allowing the ghosts to move for a random number of frames before Ms. Pacman can start moving.

The environment provides only pixels, but objects in Ms. Pacman can be identified based on their colors. Each object has an `(x,y)` location that approximates its center. Since the typical practice in Atari games is to repeat actions for four frames each, any moving object has a velocity, which is expressed as `UP`, `DOWN`, `LEFT`, or `RIGHT`. The object types and event templates are evident from these hypotheses, which the RfD agents generate for Ms. Pacman:

arrives(Pacman, Power) ⟶ Edible
catches(Pacman, Edible) ⟶ `SUCCESS`
catches(Pacman, Edible) ⟶ Eyes
collides(Pacman, Ghost) ⟶ `FAILURE`

Demonstrations for this task could vary widely. I performed a short one, taking Ms. Pacman directly to the lower left pellet and keeping her in that corner, where one edible ghost soon stumbles into her, and a bit later a regular ghost finds her. Unlike my demonstrations for other tasks, this one provides the basis for a complete causal model, but for only a partial map of the regions (which are the 36 corridors). The RfD agents discover about 60% of the regions and about 80% of the transitions as they train.

Figure 5 shows that navigation-intensive problems are approachable via RfD. The RfD agents learn to average about 8 successes per attempt within a few hundred thousand actions. Since most agents in Ms. Pacman focus on maximizing points across multiple levels, comparisons to previous results are not necessarily meaningful. However, one agent that was heavily engineered to maximize its score on the first level of Ms. Pacman averaged about 3800 points at its asymptote [34], while at their asymptotes, the RfD agents average about 5600 points on that level. Causal modeling allows them to pursue large rewards in more deliberate ways.

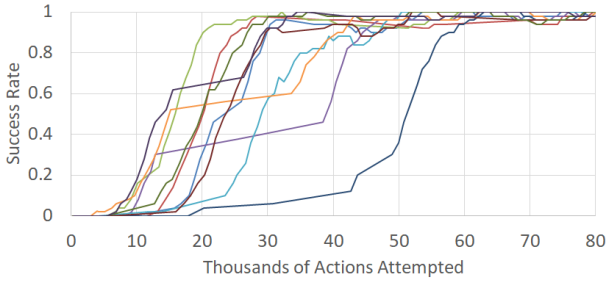

Figure 4: Learning curves for ten RfD agents in Courier. Curves are 500 attempts long and smoothed over a 50-attempt window.

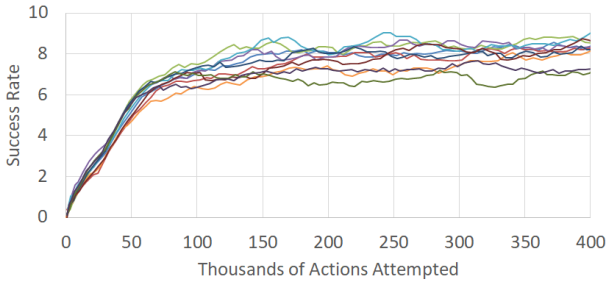

Figure 5: Learning curves for ten RfD agents in Ms. Pacman. Curves are 1000 attempts long and smoothed over a 100-attempt window.

5.3 Montezuma's Revenge

The first-room task in Montezuma's Revenge provides `SUCCESS` feedback when Joe exits the room and `FAILURE` feedback when he loses a life. In both cases, the task ends.

Regions correspond to the visual elements of the terrain: the platforms, ladders, floor, and rope. Objects are identified based on their colors, and each object has an (x, y) location that approximates its center. Moving objects have velocities that are expressed as (Δ_x, Δ_y) tuples. The object types and event templates are evident from these hypotheses, which the RfD agents generate for Montezuma's Revenge:

arrives(Joe, Key) ⟶ Door+Key
arrives(Joe, Door+Key) ⟶ `SUCCESS`
collides(Joe, Skull) ⟶ `FAILURE`
falls(Joe) ⟶ `FAILURE`

There is little variation among successful demonstrations for this task, because they must all take approximately the same route down to the key and back up to the door. To introduce some variation, the start of each game is randomly delayed by up to 400 frames, so that Joe might reach the skull at any point in its cyclical patrol. My demonstration provides the basis for a complete map, but only a partial theory; the RfD agents must learn the causes of failure as they train.

Figure 6 shows that the RfD agents reach a success rate of about 90% in this task after about 25 million actions. Comparisons to previous results are not necessarily meaningful, because most agents in Montezuma's Revenge are permitted to use all five lives. Since the skull disappears after one collision, even having two lives to spend makes it much easier to exit the room.

One approach called Deep Abstract Q-Networks does consider the single-life condition [27]. Using hierarchical deep RL with human-engineered abstract states, it rises to a peak average of approximately 300 points in approximately 50 million frames. These results seem comparable to the RfD results. The similarity suggests that when learning sparse-reward tasks, it may be less important to use a sophisticated RL algorithm than it is to acquire, in some way, an effective decomposition.

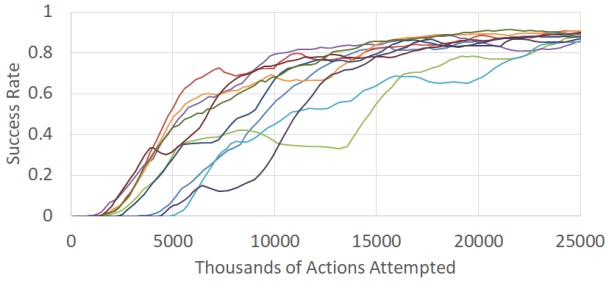

Figure 6: Learning curves for ten RfD agents in Montezuma's Revenge. Curves are 100,000 attempts long and smoothed over a 10,000-attempt window.

5.4 Extended actions

The task in Montezuma's Revenge is only slightly longer than the other tasks examined in this paper. The difference is far too small to explain the steep increase in learning time. Why is Montezuma's Revenge so hard to learn, even for agents who are well-equipped to handle sparse rewards? One recent study notes that the environment contains many dead ends, or states from which an agent will certainly lose a life, despite continuing to act for a while before it occurs [15]. For example, if Joe steps off the top platform, he falls to his death, and although the emulator continues to accept actions on the way down, none of them have any effect.

From an RL agent's perspective, this problem can be characterized in a different way: actions in Montezuma's Revenge have unpredictable durations. Normally each action lasts for four frames, but if an action begins a jump or a fall, it really lasts until Joe returns to a surface again. Humans understand this fact intuitively, since our experience in the physical world makes us expect to be powerless to change our trajectory in mid-air. Because of our intuition, we perceive the actions in this environment differently than RL agents do.

Figure 7 shows that this difference in perception has a dramatic effect on the learning process. If the actions in Montezuma's Revenge are extended until Joe stops falling, to correspond with human perceptions, the RfD agents can reach the same asymptote in about 2% of the time. The knowledge that RfD agents develop is insufficient for them to make this adjustment autonomously, so these results should not be considered achievements of RfD. They are included here just to clarify the difficulty of Montezuma's Revenge, which is due not only to the nature of its rewards, but also to the nature of its actions.

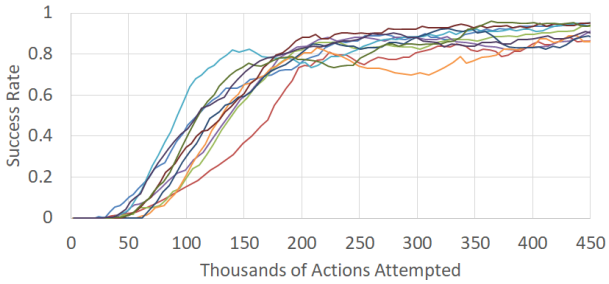

Figure 7: Learning curves for ten RfD agents in Montezuma's Revenge with extended actions. Curves are 2000 attempts long and smoothed over a 200-attempt window.

6 CONCLUSIONS

This paper proposes a framework called Reasoning from Demonstration. It is inspired by the way that humans seem to approach sparse-reward tasks with object-oriented causal reasoning. Agents in this framework acquire causal knowledge from demonstration and use it to decompose tasks effectively and practice them deliberately. The paper presents experimental evidence that in sparse-reward tasks, RfD approaches have the potential for much faster learning than traditional LfD approaches.

Another potential benefit of RfD agents is their natural resilience to low-quantity and low-quality demonstrations. One successful demonstration is often enough to establish the causal dynamics of a task. And as long as a demonstration ultimately reveals those causal dynamics, it can be arbitrarily sub-optimal without having any detrimental effect on RfD agents.

The agent described in this paper implements the RfD concept using simple components and procedures. Many of them could clearly be improved by other agents within the RfD framework. For example, richer representations of objects and policies would allow for finer control, and a more sophisticated treatment of causality would produce more robust reasoning.

Most importantly, although the agent in this paper is assisted in perceiving objects, events, and regions, future RfD agents could approach these problems autonomously. In fact, the main conclusion of this paper is that research on perception should be better integrated into research on learning in sparse-reward tasks. Structured perception is necessary for causal modeling, which clearly has potential benefits for sparse-reward learning.

This work joins ongoing discussions about combining statistical and symbolic AI [10] and about making agents learn more like humans do [22]. These discussions seem particularly relevant to the area of LfD, which is already human-inspired.

REFERENCES

[1] Pierre-Luc Bacon, Jean Harb, and Doina Precup. 2017. The Option-Critic Architecture. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*.

[2] Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew Taylor, and Ann Nowé. 2015. Reinforcement Learning from Demonstration Through Shaping. In *Proceedings of the 24th Joint International Conference on Artificial Intelligence*.

[3] Vivian Chu, Baris Akgun, and Andrea Thomaz. 2016. Learning Haptic Affordances from Demonstration and Human-Guided Exploration. In *Proceedings of the IEEE Haptics Symposium*.

[4] Luis Cobo, Charles Isbell, and Andrea Thomaz. 2012. Automatic Task Decomposition and State Abstraction from Demonstration. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*.

[5] Luis Cobo, Charles Isbell, and Andrea Thomaz. 2013. Object Focused Q-learning for Autonomous Agents. In *Proceedings of the 12th International Conference on Autonomous Agents and Multi-agent Systems*.

[6] Luis Cobo, Peng Zang, Charles Isbell, and Andrea Thomaz. 2011. Automatic State Abstraction from Demonstration. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*.

[7] Michael Dann, Fabio Zambetta, and John Thangarajah. 2019. Deriving Subgoals Autonomously to Accelerate Learning in Sparse Reward Domains. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*.

[8] Peter Dayan and Geoffrey Hinton. 1992. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems 5*.

[9] Thomas Dietterich. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research* 13, 1 (2000), 227–303.

[10] Thomas Dietterich. 2019. (10 2019). https://medium.com/@tdietterich/what-does-it-mean-for-a-machine-to-understand-555485f3ad40

[11] Edsger Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1 (1959), 269—-271.

[12] Carlos Diuk, Andre Cohen, and Michael Littman. 2008. An Object-oriented Representation for Efficient Reinforcement Learning. In *Proceedings of the 25th International Conference on Machine Learning*.

[13] Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas Griffiths, and Alexei Efros. 2018. Investigating Human Priors for Playing Video Games. In *Proceedings of the 35th International Conference on Machine Learning*.

[14] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. 2001. Relational Reinforcement Learning. *Machine Learning* 43, 1 (2001), 7–52.

[15] Mehdi Fatemi, Shikhar Sharma, Harm van Seijen, and Samira Ebrahimi Kahou. 2019. Dead-ends and Secure Exploration in Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning*.

[16] Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. 2017. Multi-Level Discovery of Deep Options. *CoRR* abs/1703.08294 (2017).

[17] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. 2016. Towards Deep Symbolic Reinforcement Learning. *arXiv* 1609.05518 (2016).

[18] D. Grollman and A. Billard. 2012. Robot Learning from Failed Demonstrations. *International Journal of Social Robotics* 4 (2012), 331–342.

[19] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. 2018. Deep Q-learning From Demonstrations. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

[20] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. 2012. Robot Learning from Demonstration by Constructing Skill Trees. *International Journal of Robotics Research* 31, 3 (2012), 360–375.

[21] Tejas Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In *Advances in Neural Information Processing Systems 29*.

[22] Brenden Lake, Tomer Ullman, Joshua Tenenbaum, and Samuel Gershman. 2017. Building Machines that Learn and Think Like People. *Behavioral and Brain Sciences* 40 (2017).

[23] Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. 2019. SDRL: Interpretable and Data-Efficient Deep Reinforcement Learning Leveraging Symbolic Planning. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*.

[24] Marlos Machado, Marc Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. 2018. Revisiting the Arcade Learning Environment. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level Control through Deep Reinforcement Learning. *Nature* 518, 7540 (2015), 529–533.

[26] Andrew Ng, Daishi Harada, and Stuart Russell. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the 16th International Conference on Machine Learning*.

[27] Melrose Roderick, Christopher Grimm, and Stefanie Tellex. 2018. Deep Abstract Q-Networks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*.

[28] Tim Salimans and Richard Chen. 2018. Learning Montezuma's Revenge from a Single Demonstration. In *Proceedings of the NeurIPS Deep Reinforcement Learning Workshop*.

[29] Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. 1992. Learning to Fly. In *Proceedings of the 9th International Workshop on Machine Learning*.

[30] Stefan Schaal. 1997. Learning from Demonstration. In *Advances in Neural Information Processing Systems 9*.

[31] Richard Sutton and Andrew Barto. 2018. *Reinforcement Learning: An Introduction* (2 ed.). MIT Press, Cambridge, MA.

[32] Richard Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence* 112, 1 (1999), 181 – 211.

[33] Matthew Taylor, Halit Bener Suay, and Sonia Chernova. 2011. Integrating Reinforcement Learning with Human Demonstrations of Varying Ability. In *Proceedings of the 10th International Conference on Autonomous Agents and Multi-agent Systems*.

[34] Lisa Torrey and Matthew Taylor. 2013. Teaching on a Budget: Agents Advising Agents in Reinforcement Learning. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*.

[35] Harm van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. 2017. Hybrid Reward Architecture for Reinforcement Learning. In *Advances in Neural Information Processing Systems 30*.

[36] Alexander Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. 2016. FeUdal Networks for Hierarchical Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*.

[37] Christopher Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (1992), 279–292.

[38] Gui Xue, Zhonglin Lu, Irwin Levin, Joshua Weller, Xiangrui Li, and Antoine Bechara. 2009. Functional Dissociations of Risk and Reward Processing in the Medial Prefrontal Cortex. *Cerebral Cortex* 19, 5 (2009), 1019–1027.

[39] Peng Zang, Peng Zhou, David Minnen, and Charles Isbell. 2009. Discovering Options from Example Trajectories. In *Proceedings of the 26th International Conference on Machine Learning*.