The Dissertation Committee for William Bradley Knox certifies that this is the approved version of the following dissertation:

Learning from Human-Generated Reward

Committee:

Peter Stone, Supervisor

Dana Ballard

Cynthia Breazeal

Bradley C. Love

Raymond J. Mooney

Learning from Human-Generated Reward

by

William Bradley Knox, B.S.; M.S.C.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2012

To my sister and my brother, with solidarity

Acknowledgments

What I write here is far from comprehensive, but I present these words as a sample of the diverse relationships and experiences that cumulatively provoke immense gratitude and a nostalgia for the present.

When I originally contacted Peter Stone, I'd chosen him based on a comparison of his website to others at UT. You'd think that level of analysis could get me into trouble, but I lucked out. Peter listens, he grants autonomy and advises rather than dictates, and he consistently shows respect for his colleagues and collaborators. Peter is reliable. He is considerate of his students' time, and he is resolutely committed to academic integrity. He is available to give feedback and is open to disagreement. Peter is also a deeply, genuinely good person. Twice I've come to him in crisis; each time, he listened and *then* gave emotionally intelligent advice. As I increasingly take on advisory roles, I dearly hope I can approximate what Peter has been for me.

I've had many other transformational teachers and mentors over the years. Before graduate school, Willard Volding, Jack Nation, Scott Austin, John McDermott, Mike Scott, and Calvin Lin all left deep and lasting impressions. At UT, Greg Kuhlmann patiently guided me through my first research project, feeding me the ideas that brought us a Robocup championship—which I suspect was the largest factor in my acceptance as a PhD student at UT. Juhyun Lee was an understanding partner and contributor of the main technical depth of our successful Robocup

@Home entry. Shivaram Kalyanakrishnan generously met with me one-on-one to discuss my reading of the Sutton and Barto text. In my research group my interests are anomalous, and the arrival of Ian Fasel as a postdoc was a stroke of good fortune. I've benefited immensely from his ideas and wide knowledge of work that relates to this dissertation. Matt Taylor has been a role model for, among other things, his superhuman efficiency and his ability to play the academic game with skill and cheer. Mohan Sridharan taught me the valuable lesson that people get annoyed when you ask for help too often, especially when they're watching Looney Tunes. And Todd Hester, perhaps the best proofreader in our lab (sorry, everyone), has improved many a submission of mine. The rest of the original LARG crew—Kurt Dresner, Dan Stronger, Nick Jong, Shimon Whiteson, David Pardoe, and Nate Kohl—those who came and went, and those who remain (and are racking up an incredible record at Robocup), you have all impacted me greatly. Chiu Au has been my officemate and friend for four years. For the last two, this officemating was only possible because Chiu generously allowed the corner of his new office to be my desk, saving my distractible mind from the hum and chatter of the robot basement. What sanity I've maintained and what productivity I've managed, I owe to Chiu.

I also owe thanks to my various collaborators outside of LARG. In the UT Psychology department, Ross Otto, Brad Love, Brian Glass, and Todd Maddox granted me the opportunity to do fascinating psychological research, bringing me full circle to the focus of my undergraduate studies. Ole Mengshoel at NASA Ames and Tal Tversky at Apple exuded calm confidence in their intern and added greatly to my research perspective. Cynthia Breazeal let me crash her lab last summer and has hired me to join her warm and inspiring Personal Robots Group family.

All of the members of PRG made that summer in Boston memorable. Specifically, Adam Setapen—a good friend and once collaborator—helped me immediately feel at home. His enthusiasm for research (including mine) is infectious and always appreciated. My meta-research discussions with Nick DePalma have been unusually memorable. Siggi Örn is the type of guy who you know is better than you in every significant way—looks especially—yet is deeply humble and consistently selfless. His help with Nexi's code saved me a tremendous amount of time. Also, I used to think that women don't let you know when they're checking you out; jogging with Siggi showed me that female furtiveness is not universal. Also in Boston but not in PRG is George Konidaris. George was my hotelmate at my first conference; in the last 4 years he's become a great friend and mentor. Our conversations about my career and reinforcement learning—especially with regards to discounting—have been invaluable.

I have also been shaped by extended interactions over many years with other researchers, who have each listened to my ideas and shared their thoughts: Sonia Chernova, Dan Grollman, Sarah Osentoski, Chad Jenkins, Eli Kim, Andrea Thomaz, Julie Shah, Nick Roy, Rich Sutton, Raj Rao, Jonathan Pillow, Michael Domjan, Dana Ballard, and Ray Mooney.

Individually and collectively, my friends and mentors in Capital City Toastmasters and at Coldtowne Theater have given me critical skills and valued memories.

Last week I moved out of my house of five years and defended this dissertation three days later. These milestones weigh heavy. *I love this town.* My seven and a half years in Austin have easily been the happiest and most fulfilling of my recollectable life. I'm so grateful for that. To everyone who is a part of this rich social web on which I've thrived, you mean so much to me.

My family has been supportive throughout my education. I love it when my cousins and uncles nerd out about AI and ask for my opinion, which I give gladly and authoritatively. I'd like to thank my grandmother Oowa (named by my cousin as a toddler) for her financial support during my post-undergrad and pre-graduate coursework, without which I would not be where I am. My recently passed step-grandfather, Bill Spiller, is my model for an intellectually rich retirement. I sorely miss his conservative perspective in the email debates we used to have. My long-deceased grandfather, Jack Knox, was the previous career academic of the family. His accomplishments set the standard against which I measure myself. I've got a ways to go.

This thesis is dedicated to my brother, the family tinkerer, and my sister, my first experimental subject. I was reminded just the other night of how much we laugh at the exact same, twisted things. I love you two and forgive you for being libertarian.

I've moved countless large items (furniture, etc.) with my dad in my lifetime. His lack of specificity often got me in trouble. Dad: "Go left." 12-year-old me: "Your left or my left?!" [By this time, he's already frustrated at my inability to coordinate with the motion he's already started.] My dad has also always wanted me to become a doctor. Oh, you meant medical, Dad? Maybe you should be more specific next time. Seriously, thanks for your unwavering faith in your children's potentials. (And for bringing Alek and Lynnette into the fold.)

When I was a kid, I used to get frustrated by the size of my homework. My mom would set a timer, for just a few minutes, and challenge me to get through some small amount in the time allotted. 20 years later, while writing this dissertation, the only way I could make consistent progress was to set a timer for 5–10 minutes and plan to do something really small. Thanks, Mom.

William Bradley Knox

The University of Texas at Austin
December 2012

Learning from Human-Generated Reward

Publication No. ____________________

William Bradley Knox, Ph.D.
The University of Texas at Austin, 2012

Supervisor: Peter Stone

Robots and other computational agents are increasingly becoming part of our daily lives. They will need to be able to learn to perform new tasks, adapt to novel situations, and understand what is wanted by their human users, most of whom will not have programming skills. To achieve these ends, agents must learn from humans using methods of communication that are naturally accessible to everyone. This thesis presents and formalizes interactive shaping, one such teaching method, where agents learn from real-valued reward signals that are generated by a human trainer. In interactive shaping, a human trainer observes an agent behaving in a task environment and delivers feedback signals. These signals are mapped to numeric values, which are used by the agent to specify correct behavior. A solution to the problem of interactive shaping maps human reward to some objective such that maximizing that objective generally leads to the behavior that the trainer desires.

Interactive shaping addresses the aforementioned needs of real-world agents. This teaching method allows human users to quickly teach agents the specific behaviors that they desire. Further, humans can shape agents without needing programming skills or even detailed knowledge of how to perform the task themselves. In contrast, algorithms that learn autonomously from only a pre-programmed evaluative signal often learn slowly, which is unacceptable for some real-world tasks with real-world costs. These autonomous algorithms additionally have an inflexibly defined set of optimal behaviors, changeable only through additional programming. Through interactive shaping, human users can (1) specify and teach desired behavior and (2) share task knowledge when correct behavior is already indirectly specified by an objective function. Additionally, computational agents that can be taught interactively by humans provide a unique opportunity to study how humans teach in a highly controlled setting, in which the computer agent's behavior is parametrized.

This thesis answers the following question. *How and to what extent can agents harness the information contained in human-generated signals of reward to learn sequential decision-making tasks?* The contributions of this thesis begin with an operational definition of the problem of interactive shaping. Next, I introduce the TAMER framework, one solution to the problem of interactive shaping, and describe and analyze algorithmic implementations of the framework within multiple domains. This thesis also proposes and empirically examines algorithms for learning from both human reward and a pre-programmed reward function within an MDP, demonstrating two techniques that consistently outperform learning from either feedback signal alone. Subsequently, the thesis shifts its focus from the agent to the trainer, describing two psychological studies in which the trainer is manipulated by either changing their perceived role or by having the agent intentionally misbehave at specific times;

we examine the effect of these manipulations on trainer behavior and the agent's learned task performance. Lastly, I return to the problem of interactive shaping, for which we examine a space of mappings from human reward to objective functions, where mappings differ by how much the agent discounts reward it expects to receive in the future. Through this investigation, a deep relationship is identified between discounting, the level of positivity in human reward, and training success. Specific constraints of human reward are identified (i.e., the "positive circuits" problem), as are strategies for overcoming these constraints, pointing towards interactive shaping methods that are more effective than the already successful TAMER framework.

Contents

Chapter 1

Introduction

A central but largely unfulfilled goal of artificial intelligence (AI) is to deploy computational agents to tackle real-world problems, making decisions that affect our lives. Though early predictions of such wide-spread deployment were overly optimistic, in recent years the prevalence of agents in the lives of the general public has noticeably increased. As I take stock now in 2012, a prime example is iRobot's Roomba, an autonomous robotic vacuum, which surpassed the five million sales mark in 2010.[1]. And despite its $395 price tag, the personal entertainment robot Pleo by Ugobe sold more than 100,000 units.[2] In addition, robotic kits such as Lego Mindstorms NXT are frequently used in classrooms to teach basic programming. The warehouse-management robots of Kiva Systems, a company bought by Amazon.com for $775 million, both coordinate and act individually to stock items strategically by their demand and retrieve these items when ordered.[3] Non-robotic agents—though they receive less media attention—are also widely used to suggest

[1] http://www.roboticstrends.com/personal_robotics/article/irobot_achieves_milestone_in_home_robot_market

[2] http://www.roboticstrends.com/personal_robotics/article/ugobe_maker_of_celebrated_pleo_goes_broke/

[3] http://spectrum.ieee.org/robotics/robotics-software/three-engineers-hundreds-of-robots-one-warehouse/0

items on retail websites, as non-player characters in video games, and so on.

Further, indications of looming agent ubiquity are easily found. The industrial robots market is projected by Daiwa Capital Markets to reach a value of $41 billion by 2020.[4] And the International Federation of Robotics forecasts that the sales of domestic service robots (e.g., for vacuuming and lawn-mowing) will total $4.3 billion in 2014 and that sales of "entertainment and leisure" robots will be $1.1 billion in 2014.[5] Industry giants Honda, Toyota, Microsoft, and Intel are all actively researching personal robotics. Microsoft co-founder Bill Gates, writing in a 2008 issue of Scientific American, argued that the personal robots industry will soon experience explosive growth similar to the emergence of personal computers in the 1970s. And Google has recently deployed seven robotic Toyota Priuses, which have together amassed over 140,000 miles with "only occasional human control" as of October 2010.[6] The state of Nevada had passed laws that are friendly to robot cars, and BMW, Volkswagon, and Audi are also developing robot cars.[7] Additionally, low-cost personal robots have recently become available. For instance, the $149 mobile base Romo attaches to a iOS or Android phone, creating a robot that uses the phone's sensors (e.g., camera and microphone), screen, and speakers to interact with its environment; the phone also provides the computation for the robot.[8]

In addition to indications of increasing deployment of computational agents, the amount of interaction with humans should increase as well. In 2011, U.S. President Barack Obama announced funding by the National Science Foundation for research on human-robot collaboration. And in the coming months, Rethink Robotics—a startup by iRobot co-founder Rod Brooks—will unveil its industrial

[4] `http://www.nytimes.com/2012/04/14/business/global/kuka-german-maker-of-robots-will-expand-in-china.html`

[5] `http://www.ifr.org/service-robots/statistics/`

[6] `http://www.nytimes.com/2010/10/10/science/10google.html`

[7] `http://www.forbes.com/sites/daviddisalvo/2012/05/07/googles-robot-cars-get-green-light-to-hit-the-road/`

[8] `http://www.ohgizmo.com/2012/07/25/meet-romo-the-smartphone-robot/`

robot product, which "can be taken out of the box, taught a task by an untrained factory worker, and be productive in a few hours, eliminating the need for systems integration. They are safe to interact with people at close range and are easy to train and retrain on the fly."[9]

As computational agents migrate from research labs to places of industry, our homes, and our driveways, they will need to be able to learn new skills and adapt to their specific environments. For some learning tasks, the agents will have preprogrammed objective functions and be able to learn without significant cost (in the form, for example, of machine wear, property damage, or injury). However, (1) there will be tasks for which correct behavior is determined only by the user's preferences and consequently cannot be pre-programmed. Instances include an effective stimulation strategy for an amputee's neural prosthetic, the steps needed to do a person's laundry, and the style of assistance needed by an automobile mechanic. Additionally, (2) there will be tasks that must be learnt quickly—not from scratch—to avoid costs from repeated trials and prolonged low performance during early learning. Such tasks include dishwashing and collaborating with a human on an assembly line, both of which might use a combined measure of productivity and quality of result as an objective function.

Considering these two expected demands on real-world learning agents, it is increasingly important that people, without programming skills or long-term instruction

1. can specify and teach desired behavior and
2. share task knowledge when correct behavior is already indirectly specified by an objective function.

If methods are developed which allow humans to communicate their desires and transfer their knowledge through natural communication, both of these critical needs

[9] http://www.rethinkrobotics.com/sr_dev_relations_eng.html

can be addressed. Currently, most knowledge transfer from humans to agents occurs via programming, which is time-consuming and inaccessible to the general public. In contrast, natural forms of communication do not have these limitations. Types of natural communication that can fully specify behavior includes

- advice and instruction (likely verbal),
- demonstrations, and
- human-generated reward signals.

Of these three, behavioral specification by demonstration and by advice or instructions have received considerable attention (see Section 2.5 for a review of such work). This thesis focuses on the third type—learning from human-generated signals of numeric reward—investigating it to a depth beyond that of the sparse studies that have previously considered the problem. Here, "reward" is broadly and indefinitely defined to include communications of approval or disapproval; of judgements of good or bad behavior or outcomes; of rewards or punishments, using terminology from outside of machine learning;[10] or something similar. Similarly, human reward might be communicated in numerous ways, including by voice, facial expression, a mouse, or push-buttons.

In this thesis, I specifically focus on the case when a human is *intentionally* sending reward signals, training the agent to learn high-quality policies for sequential decision-making tasks. We call this training scenario *interactive shaping*[11] (see

[10] Following the terminology of reinforcement learning [90], I use "reward" throughout this thesis to mean signals of both positive and negative valence. The word "punishment" is only used when referring to the trainers' perspectives (since they will be unfamiliar with my usage) or to a more psychological view of reward and punishment, where the learner is biological.

[11] We use the term "shaping" similarly as in the animal learning literature (in which it was initially developed by B.F. Skinner). There, shaping is defined as training by reinforcing successively improving approximations of the target behavior [12]. In reinforcement learning literature, it is sometimes used as in animal learning, but more often "shaping" is restricted to methods that combine a supplemental, shaping reward signal and the reward signal of the task environment into a single signal [70].

Section 2.3 for a formal definition). The following subsection motivates this focus on training by human reward.

1.1 Interactive shaping

As a form of communication from human to agent, reward has unique advantages that make it a powerful basis for learning. In this section, I first argue its complementarity to other forms of natural communication. Then I similarly explain the potential of human reward to improve upon learning of tasks with predefined objective functions.

1.1.1 Learning via natural communication

Among the three forms of natural communication listed above, interactive shaping has unique advantages. Though undoubtedly useful, other communication forms have significant constraints that make human reward a potent alternative and complement.

Learning from demonstration (also called "imitation learning") requires a demonstration interface that is specific to the agent's action space; such an interface may be intractable with certain complex agents. Demonstration also requires a clear policy in the trainer's mind [4]. Instruction—or similar but less binding advice—requires a highly expressive form of communication and thus generally relies on a natural language processing interface. Natural language processing is unsolved, and such natural language interfaces have consequently only been successful for specific tasks with relatively narrow vocabularies [57, 64, 68] or limited grammatical structures [100]. Communication by human reward, in contrast, can occur through a *simple and intuitive communication interface* that can be implemented easily and is nearly universal to both trainers and tasks.

In addition to interface-level advantages of interactive shaping, intuition suggests that it also makes *less demands on the human trainer*, both in requisite expertise and attention.[12] To correctly formulate instructions or demonstrate a task, a human trainer must be an expert at the task. Criticizing the agent comes more easily; the human need only judge the outcome. Not everyone is an expert, but, as the saying goes, everyone is a critic.

Moreover, it is likely that the cognitive load of critical evaluation is less than that of both demonstrating and instruction. Cognitive load appears to be nonexistent when considering human reward that is communicated unintentionally (i.e., not reward from interactive shaping). In this case, a comparison with other forms of communication is less important, since the human is not actively choosing between forms. People constantly provide evidence of their appraisal of other social agents' behavior, through facial expression, tone of voice, body language, or other means. Such information—which could be mapped to a reward signal—is available to the agent. Without an understanding of how to use appraisal cues to guide its own behavior, an agent is wasting this information. Though I do not explicitly address unintentionally communicated reward signals in this thesis, I believe that the conclusions herein—especially as informed by the problems of reward positivity (explored in Chapter 6)—will help inform how to learn from such unintentional signals.

Though interactive shaping has clear advantages that distinguish it from other natural methods of communication, an ideal learning agent would harness multiple forms of communication. For instance, a trainer might demonstrate behavior first and then use reward-based feedback to fine-tune the behavior. By restricting the scope of this thesis to this single communication type, I aim to develop a toolset for other researchers of agent learning that can be added as a module to other agent

[12] High-level verbal commands that convey a desired goal state, on the other hand, also make low demands on the trainer. However, such methods require a sub-algorithm that plans to reach the goal or learns from experience to reach the goal, using what understanding the trainer has of good behavior less than the other forms of natural communication discussed here.

learning algorithms without overlapping with those algorithms' evaluative input.

Despite these advantages of interactive shaping, few researchers have addressed the topic, and none have made it the object of a dissertation-length investigation focus. This thesis deeply explores the potential to learn from human reward.

1.1.2 Tasks with predefined objective functions

Reinforcement learning [90, 54] (RL) is the dominant framework for agent learning when an objective function is predefined. In RL, agents use periodic reward signals to learn the quality of behavioral policies and improve upon those policies. The objective function, which outputs evaluations of behavioral quality, is a time-discounted sum of such reward signals. Reinforcement learning algorithms have achieved a number of notable successes [102, 54, 1] and continue to improve. RL is an essential framework for learning, but current RL algorithms fall short of meeting the needs of socially embedded, real-world agents.

First, for some problems, current RL algorithms learn too slowly without informative prior knowledge. Early learning trials often incur a large loss of potential reward. For real-world tasks with real-world costs, the agent will need to quickly reach a good policy (though not necessarily an optimal one). Fortunately, injecting task knowledge from outside sources can potentially improve initial performance, total cost, and final performance [96]. Through natural teaching methods, a human teacher can transfer his or her task knowledge, potentially improving learning and achieving better performance throughout a learning session.

Secondly, RL algorithms inflexibly define the set of optimal behaviors by a predefined reward function (and transition function), which typically require programming to change. In contrast, natural teaching methods such as shaping allow the human trainer to define—and redefine—correct behavior in real time, during the teaching session. This flexibility allows the human user to both specifically customize the agent's behavior to his or her needs and even change the target behavior mid-training, as might happen, for example, when the human learns more about the task through the training process.

Thus, agents that can learn from shaping and other natural teaching methods have important advantages over current RL algorithms, making the natural methods viable alternatives to learning from predefined reward functions. On the other hand, these methods also have disadvantages—chiefly, the cost of human time during training and the myriad suboptimalities of a human teacher—so they can be seen as complementary to learning from an explicitly defined reward function. Considering these complementary strengths and weaknesses, this thesis not only examines how to learn exclusively from human reward (Chapters 3 and 6), but it also considers how to learn from both the human and an encoded evaluation function—when available—using shaping to enhance RL algorithms (Chapter 4).

In addition to its role as a complementary approach to interactive shaping, reinforcement learning is critical in this thesis as a general framework with which I compare our various approaches to the problem of interactive shaping. To familiarize the reader with reinforcement learning, I provide a longer introduction to RL in Section 2.1.

1.2 Thesis question and contributions

With the previous motivation in mind, this thesis focuses on this question:

> **How and to what extent can agents harness the information contained in human-generated signals of reward to learn sequential decision-making tasks?**

In addressing this question, this thesis yields these core contributions:

1. **Problem definition of interactive shaping.** The general problem of learning from human reward is previously undefined. This thesis gives an operational definition of what we term the ”Interactive Shaping Problem” in Section 2.3.

2. **The TAMER framework for learning only from human reward.** In Chapter 3, this thesis introduces the TAMER framework, which directly addresses the thesis question, providing algorithms for task learning when human reward is the only source of evaluative information available to the learning agent. Included in the framework are directives for learning a predictive model of human reward, accounting for delay in the human trainer’s evaluation and communication of the reward, and using the learned model myopically for action selection. Positive empirical results for implementations of the TAMER framework in three domains are presented.

3. **Learning from both human and predefined reward.** Sometimes both a human trainer and an encoded reward function will be available to a learning agent. In Chapter 4, this thesis empirically examines numerous plausible techniques for learning from both evaluation signals, specifically focusing on how to combine a learned (and possibly changing) model of human reward with temporal difference reinforcement learning algorithms.

4. **Psychological studies of human trainers.** In addition to asking how to create the best human-shapable agent, this thesis investigates how humans shape the agents in Chapter 5. Little work has been done on how people teach via reward, making this a fruitful line of investigation. In two relatively large, well-controlled experiments, we investigate how the trainer’s feedback is impacted by the trainer’s self-perceived role and by agent misbehavior. These experiments provide early exemplars of the emerging technique of using com-

putational *learning* agents as highly specifiable social entities in experiments on human behavior.

5. **An investigation of reward discounting in interactive shaping.** Chapter 6 returns to the problem of interactive shaping, examining a critical difference between the TAMER framework and other work on learning from human reward: how the value of future reward is discounted. Focusing on goal-based tasks in episodic domains, we identify what we call the positive circuits problem that arises from the bias towards positivity in human-generated reward. Using data from a TAMER experiment and a new user study, we show that TAMER's myopic discounting effectively avoids the positive circuits problem. In a further user study, we reveal another technique for addressing this problem—"tricking" the agent into thinking its task is continuing, not episodic—which may ultimately lead to algorithms that differ from TAMER but are more powerful for interactive shaping.

1.3 Reader's guide

Here I give brief recommendations for potential strategies in reading this thesis. The thesis consists of five chapters bookended by the introductory Chapter 1 and the concluding Chapter 7.

Chapter 2 gives crucial background information on reinforcement learning (Section 2.1) and regression (Section 2.2). Readers who are already comfortable with these topics may wish to skim these sections and use them for reference as needed. In Section 2.3, I formally specify interactive shaping, the central problem addressed by this thesis; consequently, Section 2.3 is critical reading for a deep understanding of the thesis' contributions. Section 2.4 is meant as a reference for notation and terminology that is introduced herein. Lastly, Section 2.5 gives an overview of previous work on learning tasks from human reward, followed by a summary of past research that addresses the broader problem of learning tasks from human instruction.

Chapters 3–6 constitute the core technical chapters of the thesis and are described in the previous section (1.2). An understanding of Chapter 2 is essential for reading Chapter 3 on the TAMER framework, which in turn is the foundation for Chapters 4, 5, and 6. These latter three chapters can however be read independently of each other. Additionally, I provide a short summary of interactive shaping and the TAMER framework in Appendix A, which is intended to serve as a minimally adequate substitute for Chapters 2 and 3 for readers who are interested in one or more later chapters but do not have time to read the full versions of the two prerequisite chapters.

The conclusion, Chapter 7, reviews the five technical contributions of the thesis, chapter by chapter. From these contributions, I discuss which approaches to interactive shaping are currently most effective and which are most promising as directions for further research. In Section 7.1, I then provide a list of potential near-term projects that could extend this thesis, further improving the effectiveness of interactive shaping. Lastly, in Section 7.2 I present a broader vision for research on algorithms that learn from human users.

Chapter 2

Background and Problem Definition

This chapter prepares for and motivates the core chapters of this thesis(3–6). There are many detailed treatments of the categories of algorithmic learning used in this thesis, specifically reinforcement learning [5, 39, 90, 91] and supervised learning [67, 7]. This thesis assumes prior familiarity with these learning categories; the ambition of this chapter is only to review them and to highlight their most relevant points for the purpose of understanding the thesis.

I first introduce reinforcement learning. In the coming chapters, the various algorithms I propose are usually integrated with, compared to, or are reinforcement learning algorithms. Though not all of our novel algorithms are reinforcement learning, reinforcement learning provides the principle framework in which I situate the contributions of this thesis. After describing reinforcement learning, I briefly describe supervised learning, which is also used as an important component of our algorithms.

I then define the problem of *interactive shaping*. Interactive shaping can be roughly described as learning from human reward. It is the focus of this thesis. Lastly, the chapter contains a reference for the notation that I create and use throughout the remainder of the thesis.

2.1 Reinforcement learning

Reinforcement learning (RL) is a set of methods for learning how to perform tasks. In RL, an agent takes actions that affect the state of its environment, and the quality of its actions, given state, affects the amount of reward the agent receives from its environment. A principle challenge of reinforcement learning is that the agent seeks to receive as much cumulative reward as possible—often with future reward discounted in worth—so the quality of an individual action can only be evaluated in the context of both its effect on future state and the actions that occur after the action of interest.

Early research on reinforcement learning drew inspiration from the animal learning literature on classical and operant conditioning [90], where the learning of animals—including humans—is modeled to occur as the consequence of events that naturally reward or punish the animal and of the environmental cues that add context to such events [12]. These environmental cues are analogous to environmental state in RL, and reward and punishment in operant conditioning are together analogous to reward in reinforcement learning. In RL, reward signals can have any real value, positive or negative; the term “punishment” is not used widely in reinforcement learning.

2.1.1 Markov decision processes

Reinforcement learning (RL) [90] with fully observable state usually concerns solving tasks formulated as Markov decision processes (MDPs), denoted as $\{S, A, T, D, R, \gamma, \}$. Here, S and A are the sets of possible states and actions. Time is divided into discrete steps, during which an agent observes the current environmental state, takes an action that affects the state, and observes the next state at the start of the next time step. An environmental state and the action that is taken in it are often referenced together as a *state-action pair.* $T : S \times A \times S \rightarrow \mathbb{R}$, called the transition function, describes the probability of transitioning from one state to another given a specific action. D is a probabilistic distribution over states, from which the starting state is chosen.

$R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, which outputs a numeric reward at each step, the value of which is dependent on the state-action pair and resultant next state. γ, the discount factor, controls the comparative worth of reward at various points in the future; a reward r received n steps after the present is worth $\gamma^n r$ reward in the present. An agent's prescription for behavior can be described by a *policy*, $\pi : S \times A \rightarrow \mathbb{R}$, where $\pi(s, a) = P(a|s)$, the probability of the agent choosing action a in state s. A deterministic policy is more simply denoted as $\pi : S \rightarrow A$.

The *return* of a trajectory of experienced state-action pairs from times $t = 0, 1, ...n$ is $\sum_{t=0}^{n} [\gamma^t R(s_t, a_t)]$. RL algorithms for control seek to learn policies that *maximize expected return* from each state-action pair over an infinite horizon. Expected return for a policy π is denoted as $Q^\pi(s, a)$, where $\sum_{t=0}^{\infty} E_\pi[\gamma^t R(s_t, \pi(s_t))] = Q^\pi(s, a)$, the expected sum of discounted rewards when starting from state s, taking action a, and interminably following policy π thereafter. The set of optimal policies for an MDP consists of all π^* such that $Q^{\pi^*}(s, \pi^*(s)) \geq Q^\pi(s, \pi(s))$ for any π for all $s \in S$. $Q^\pi(s, a)$ is referred to as the *value* of the state-action pair (s, a) given π. Though not often used in this thesis, one can also refer to the value of a state, $V^\pi(s) = max_a(Q^\pi(s, a))$. V and Q are respectively called a *state-value function* and an *action-value function*, and they are both considered *value functions.*

All of the experimental task domains in this thesis have fully observable state. I consider problems with hidden state in the discussion of future research directions in Chapter 7.

2.1.2 Categories of reinforcement learning

Reinforcement learning problems and the algorithms that address them can both be differentiated by many characteristics. A few pertinent categorical divisions are described in this section.

Continuing and episodic tasks

Tasks defined as MDPs are either *continuing* or *episodic.* A continuing task begins in a state drawn from the start state distribution D and continues indefinitely; consequently, any reward can be attributed to any past action. An episodic task consists of recurrent episodes of action. An episode begins according to D and ends when an *absorbing state* is reached. Through the device of absorbing states—from which an agent cannot transition or receive non-zero reward—episodic tasks can be considered a specific type of continuing task. Though the agent theoretically continues to act forever from an absorbing state, its assessment of the return accrued by previous action is not affected, so the learning algorithm simply begins a new episode.

Value-function based RL and policy search methods

Most current reinforcement learning algorithms can be placed in one of two categories: (1) those that learn value functions and derive their policies from the value function and (2) those that learn policies directly without modeling a value function. Algorithms in this second category are called *policy search* algorithms. A policy search algorithm learns parameters for a policy, evaluating candidate policies by the performance of one or more trajectories resulting from the policy. Therefore, though policy search algorithms require an objective function that evaluates whole trajectories, the objective function need not be built from a reward function, which provides per-step feedback that can be added to evaluate a trajectory. Consequently, policy search algorithms apply to reinforcement learning problems but are not confined to them.

Function approximation

Implementations of value-function based RL algorithms can be divided between those that represent the value function with a separate value-indicating parameter for each state-action pair—so-called tabular representations—and those for which multiple state-action pairs share the same parameters—*function approximation* representations.

Extracting *features* from a state-action pair and using them as input to a Q-function is a common component of function approximation. Feature extraction occurs by $\phi : S \times A \rightarrow \mathbb{R}^n$, where the feature vector is of size n. Features often include indicator functions over multiple states (as in tile coding), radial basis functions, or hand-engineered measurements over multiple state variables (e.g., extracting the angle and distance to an opponent from absolute locations and orientations).

As an example, a common function approximator is a linear model over a vector of features. If an action-value function is being learned, such a function approximator represents it as $Q(s,a) = \overrightarrow{\theta} \overrightarrow{\phi}_{s,a}$, where $\overrightarrow{\phi}_{s,a} = \phi(s,a)$, the feature vector for the current state and action. Updates to Q affect the column vector $\overrightarrow{\theta}$ of parameters of the linear model. Interestingly, a tabular representation can technically be seen as a type of linear representation.

In general, any combination of feature extraction, value-function representation, and regression algorithm is a potential function approximator. Section 2.2 provides a short overview of regression, including incremental gradient descent over a linear model, which is used in RL algorithms in Chapters 3, 4, and 6.

Typically, function approximation is designed to reduce the parameter space of the value function. This reduction has the advantage of causing generalization between state-action pairs, where experience from one state-action pair informs the agent's evaluation of "similar" state-action pairs. Therefore, when a function approximator implicitly makes effective assumptions about the relationship between state-action pairs, the speed of learning increases. However, function approximation often prevents the agent from learning optimal policies; in the worst case, it prevents behavioral improvement altogether. Consequently, effective approximate representations are actively being researched [66, 56, 74, 73]. Often, such research focuses on the choice of *features* for a linear model representation of the value function. Features are computed from a state-action pair (and possibly the subsequent state) and replace the state-action pair as direct inputs to the value function.

2.1.3 Reinforcement learning algorithms in this thesis

In this thesis, two different reinforcement learning algorithms are used in experiments: value iteration and SARSA(λ). These algorithms are fundamental algorithms in reinforcement learning from which many state-of-the-art RL algorithms are built; we typically use the algorithms to test relative improvements with different interactions with human reward (Sections 4 and 6), where we hope that the algorithms are representative of many others.

Both algorithms are built from the *Bellman equation*,

$$Q^{\pi}(s,a) = \sum^{s' \in S} T(s,a,s')[R(s,a,s') + \gamma Q^{\pi}(s', \pi(s'))], \tag{2.1}$$

which is derived from the definition of $Q^{\pi}(s,a)$, $Q^{\pi}(s,a) = \sum_{t=0}^{\infty} E_{\pi}[\gamma^t R(s_t, \pi(s_t))]$ [90]. (Recall that $T(s,a,s') = P(s'|s,a)$.) For an optimal policy π^*, the Bellman equation can be expressed alternatively as the *Bellman optimality equation*,

$$Q^{\pi^*}(s,a) = \sum^{s' \in S} T(s,a,s')[R(s,a,s') + \gamma \; max_{a'} Q^{\pi^*}(s', a'))]. \tag{2.2}$$

Algorithm 1 Value iteration

```
Initialize $V$ arbitrarily. E.g., $V(s) = 0$ for all $s \in S^+$
1: repeat
2:   $\Delta \leftarrow 0$
3:   for all $s \in S$ do
4:     $v \leftarrow V(s)$
5:     $V(s) \leftarrow max_a \sum^{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s'))]$
6:     $\Delta \leftarrow max(\Delta, |v - V(s)|)$
7:   end for
8: until $\Delta < \theta$ (a small positive number)
9: return Deterministic policy $\pi$ such that
       $\pi(s) = argmax_a \sum^{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s'))]$
```

For a state value function V, the Bellman equation is

$$V^{\pi}(s) = \sum^{a \in A} \pi(s, a) \sum^{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^{\pi}(s')] \tag{2.3}$$

and the Bellman optimality equation is

$$V^{\pi^*}(s) = max_a \sum^{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^{\pi^*}(s'))]. \tag{2.4}$$

Value iteration

Value iteration learns a policy by repeated application of the Bellman optimality equation for state value functions. Following its presentation in Sutton and Barto [90], I define value iteration in Algorithm 1, assuming a tabular representation of V. Value iteration requires the agent to know the transition function T and the reward function R. When it is applicable, value iteration is theoretically guaranteed to converge to the optimal value function—from which an optimal policy can easily be derived—if the repeat loop starting at line 1 never terminates (i.e., $\theta = 0$).

SARSA(λ)

In contrast to value iteration, SARSA(λ) does not require full knowledge of T and R. Instead, SARSA(λ) learns from experience interacting with the environment, sampled transitions of the form $(s, a, R(s, a, s'), s', a')$. (Replacing "$R(s, a, s')$" with shorthand "r" exhibits the basis for the algorithm's name.) To explain SARSA(λ), I first describe the simpler Sarsa algorithm in detail and then give a high-level description of how SARSA(λ) extends Sarsa through eligibility traces. The Bellman equation for an action-value function (Equation 2.1) provides two methods for estimating the value of a state s given a current approximation of state value by Q and a sampled transition $(s, a, R(s, a, s'), s', a')$. The current value of $Q(s, a)$ is itself one estimate. The second estimate comes from knowledge that $E_{s,a}[R(s, a, s') + \gamma Q(s', a')] = \sum^{s' \in S} T(s, a, s')[R(s, a, s') + \gamma Q^{\pi}(s', \pi(s'))]$, the right side of Equation 2.1. Consequently, $R(s, a, s') + \gamma Q(s', a')$ is also an estimate of $Q(s, a)$. This estimate, though possibly noisy, is grounded by the added information of the experienced reward.

Considering the expectation of $E_{s,a}[R(s, a, s') + \gamma Q(s', a')]$ to be a more accurate assessment of the true value of (s, a) than $Q(s, a)$, a so-called temporal-difference error δ is calculated between the two estimates:

$$\delta = R(s, a, s') + \gamma Q(s', a') - Q(s, a) \tag{2.5}$$

The Sarsa algorithm is described for episodic tasks in Algorithm 2, again following the description from Sutton and Barto [90]. Line 6 requires an action selection mechanism; one such mechanism that is commonly used is ϵ-greedy. When acting by ϵ-greedy, at each step an agent chooses the action that is greedy with respect to its current Q-function with probability $1 - \epsilon$, and otherwise it chooses among all potential actions (including the greedy action) with equal probability. By defining its policy by its repeatedly updated Q-function, a Sarsa agent generally improves

Algorithm 2 Sarsa

```
Initialize Q arbitrarily.
repeat (for each episode)
  Initialize $s$
  Choose $a$ from $s$ using a policy derived from Q (e.g., $\epsilon$-greedy)
  repeat (for each step of episode)
    Take action $a$, observe $s'$ and $R(s, a, s')$
    Choose $a'$ from $s'$ using a policy derived from Q (e.g., $\epsilon$-greedy)
    $\delta \leftarrow R(s, a, s') + \gamma Q(s', a') - Q(s, a)$
    $Q(s, a) \leftarrow Q(s, a) + \alpha\delta$
    $s \leftarrow s'$
    $a \leftarrow a'$
  until $s$ is terminal
```

its policy over time. In fact, a Sarsa agent is guaranteed to learn an optimal policy and optimal action-value function if it visits each state-action pair an infinite number of times, it acts greedily in the limit, and α is sufficiently small.

Consider a Sarsa agent that experiences a sequence $(s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, ...)$, where $r_i = R(s_i, a_i, s_{i+1})$. The Sarsa algorithm does not use any reward after r_1, such as r_2, to update $Q(s_1, a_1)$ unless it revisits s_1. The reward r_2 would affect $Q(s_2, a_2)$, but the information gained about $Q(s_2, a_2)$ would only "trickle back" to states that might precede s_2 indirectly by revisiting those states and s_2. It is often desirable to allow learning *directly* from reward experienced in the future beyond the next transition. To this end, Sarsa and any other algorithm that learns from temporal-difference error can be extended with eligibility traces. Eligibility traces preserve a type of slowly fading memory of what states have been experienced, permitting newly experienced reward to immediately modify the estimates of previously experienced states. Sarsa with eligibility traces is called sarsa(λ) (Algorithm 3, following Sutton and Barto's definition). More thorough treatments of sarsa(λ) and eligibility traces in general can be found in previous work [90, 55].

Algorithm 3 SARSA(λ)

```
Initialize $Q$ arbitrarily and $e(s,a) = 0$, for all $s \in S$, $a \in A$.
repeat (for each episode)
  Initialize $s$
  Choose $a$ from $s$ using a policy derived from Q (e.g., $\epsilon$-greedy)
  repeat (for each step of episode)
    Take action $a$, observe $s'$ and $R(s,a,s')$
    Choose $a'$ from $s'$ using a policy derived from Q (e.g., $\epsilon$-greedy)
    $\delta \leftarrow R(s,a,s') + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    for all $s \in S$, $a \in A$ do
      $Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)$
      $e(s,a) \leftarrow \gamma \lambda e(s,a)$
    end for
    $s \leftarrow s'$
    $a \leftarrow a'$
  until $s$ is terminal
```

2.2 Regression

Supervised learning is the problem of creating a model of a process from samples of the process's input and output. When the output is one of a discrete set, supervised learning is called classification. When the output is one or more real numbers, it is called *regression*. As I first explain in Section 3.1.1, we employ regression to model human reward. Regression is also used for function approximation in reinforcement learning algorithms (see Section 2.1.2). Since classification is never applied in this thesis, I focus exclusively on regression in this section.

I now define a general and common form of regression more precisely. First assume that the process to be modeled is a deterministic function with one or more input values and a single continuous output value. In this case, a regression algorithm builds a model $\hat{f}$ that accurately approximates a target function f. As training data, the algorithm is given a set $X = \{(\overrightarrow{x}_1, y_1), (\overrightarrow{x}_2, y_2), ..., (\overrightarrow{x}_n, y_n)\}$ of samples of input and output from f. Each sample consists of a column vector of inputs $\overrightarrow{x}$ and an output y, where $f(\overrightarrow{x}) = y$. This output, also called a label, distinguishes supervised learning from unsupervised learning, in which training data do not contain output data (and the notion of "output" is often irrelevant). Once learned, $\hat{f}$ can be used to predict outputs $\hat{f}(\overrightarrow{x}_q)$ for any unlabeled query $\overrightarrow{x}_q$.

The broad goal of regression is to build an $\hat{f}$ such that $\hat{f}(\overrightarrow{x}_q) \approx f(\overrightarrow{x}_q)$ for any future query sample. However, with limitation from the size of the training set and the representational power available, a regression algorithm must make trade-offs that result in higher accuracy for certain samples than for others. Therefore, to formalize the desired trade-off, an overall accuracy measure is often designated. For example, an algorithm could attempt to minimize mean squared error (MSE) on the training set:

$$\hat{f} \leftarrow argmin_{\hat{f}} \frac{1}{|X|} \sum^{(\overrightarrow{x},y)\in X} (y - \hat{f}(\overrightarrow{x}))^2 \tag{2.6}$$

The incremental gradient algorithm described in this section finds a local minimum of MSE.

Often, f is not a function but rather outputs probabilistically from a distribution conditioned on $\overrightarrow{x}$. In such cases, $\hat{f}$ might model the conditional distribution $P(y|\overrightarrow{x})$ or the expectation of $f(\overrightarrow{x})$. Our models of human reward—introduced in Chapter 3—are designed to approximate the expectation of human reward given the input of a state-action pair. We implement two different regression algorithms to model human reward, incremental gradient descent and k-nearest neighbors.

2.2.1 Incremental gradient descent

Using a parametrized representation for $\hat{f}$, *incremental gradient descent* (also known as "stochastic gradient descent") models f by modifying the parameters of $\hat{f}$ separately for each training sample. Such an incremental approach is especially useful when the regression algorithm has a very large training set or experiences a stream of samples and needs to make predictions during training; incremental gradient descent is relatively quick in both its adjustment of $\hat{f}$ to a new sample and its answers to queries. I explain the algorithm from the perspective that samples stream into the algorithm.

Given a column vector $\overrightarrow{\theta}$ that parametrizes $\hat{f}$ and a new sample $(\overrightarrow{x}, y)$, incremental gradient descent first calculates the gradient vector for half of squared error,

$$\nabla_{\overrightarrow{\theta}} \frac{1}{2}[y - \hat{f}(\overrightarrow{x})]^2 = [y - \hat{f}(\overrightarrow{x})] \nabla_{\overrightarrow{\theta}} \hat{f}(\overrightarrow{x}) \tag{2.7}$$

$$= [y - \hat{f}(\overrightarrow{x})] < \frac{\partial \hat{f}(\overrightarrow{x})}{\partial \theta_1}, \frac{\partial \hat{f}(\overrightarrow{x})}{\partial \theta_2}, \ \dots \ , \frac{\partial \hat{f}(\overrightarrow{x})}{\partial \theta_n} >^T, \tag{2.8}$$

where θ_i is the ith component of $\overrightarrow{\theta}$ and $y - \hat{f}(\overrightarrow{x})$ is the prediction error for the sample. With this gradient, $\overrightarrow{\theta}$ is updated:

$$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \alpha \nabla_{\overrightarrow{\theta}} \frac{1}{2}[y - \hat{f}(\overrightarrow{x})]^2. \tag{2.9}$$

In this equation (2.9), α is a step-size parameter. The incremental gradient descent update improves the model by moving parameters with the gradient such that the error for each individual sample gets a little smaller.

For linear models $\hat{f}(\overrightarrow{x}) = \overrightarrow{\theta}^T \overrightarrow{x}$, so

$$\nabla_{\overrightarrow{\theta}} \frac{1}{2}[y - \hat{f}(\overrightarrow{x})]^2 = \nabla_{\overrightarrow{\theta}} \frac{1}{2}[y - \overrightarrow{\theta}^T \overrightarrow{x}]^2 \tag{2.10}$$

$$= [y - \overrightarrow{\theta}^T \overrightarrow{x}] \nabla_{\overrightarrow{\theta}} \overrightarrow{\theta}^T \overrightarrow{x} \tag{2.11}$$

$$= [y - \overrightarrow{\theta}^T \overrightarrow{x}] \overrightarrow{x}, \tag{2.12}$$

yielding the simplified update

$$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \alpha[y - \overrightarrow{\theta}^T \overrightarrow{x}]\overrightarrow{\theta}. \tag{2.13}$$

Given certain conditions on the step-size parameter α and that samples are randomly drawn from a static distribution, incremental gradient descent is guaranteed to converge to a local minimum of MSE over the distribution of samples. For linear models, incremental gradient descent converges on the global minimum, creating a model that is globally optimal with respect to MSE.

2.2.2 k-nearest neighbors

k-nearest neighbors (kNN) predicts a query's label using the k "nearest" training samples (i.e., neighbors) [67]. To determine distance between samples, some distance function d must be specified such that $d(\overrightarrow{x}_q, \overrightarrow{x}_i)$ is the distance between the query's input vector $\overrightarrow{x}_q$ and the input vector of training sample i. In this thesis, d always outputs the Euclidean distance between the two vectors. Given a distance function, kNN's prediction of a query's label is a function of the k labels of the k nearest training samples. Most commonly, the prediction is an average of the k labels, sometimes with each label weighted according to its distance from the query.

2.3 Specification of interactive shaping

In this key section, I introduce the problem this thesis focuses on, interactive shaping. Interactive shaping is roughly the problem of learning from human reward.[1]

[1] The terms "human reward" and "interactive shaping" are chosen from consideration of the meaning of similar terms in both the animal learning and the reinforcement learning literatures. Our use of "human reward" comes from other researchers referring to human-generated, scalar feedback signals as "reward" in past work [36, 105, 76, 101]; the usage of the words "reward" and "punishment" in the instructions given to subjects in our experiments; and the apparent similarity of the feedback signal to real-valued reward in reinforcement learning. Perhaps most importantly, we seek algorithms that capture the simplicity and ease of teaching by reward and punishment

Each core chapter in this thesis either investigates solutions to the problem of interactive shaping (Chapters 3 and 6) or builds from an assumed solution (Chapters 4 and 5). The Interactive Shaping problem [47] is summarized by the following question: how should an agent, given a human trainer observing the agent's behavior and delivering evaluative reward signals, be designed to leverage the human reward signals to learn good behavior? To express this problem more precisely, I define the task environment.

We assume that the agent's task environment is an MDP\R: a Markov decision process without a reward function or a discount factor γ.[2] An MDP\R is specified by the tuple (S, A, T, D). In this specification S is the set of possible states, and A is the set of possible actions. The transition function T determines the probability of transitioning from one state to another given an action, $T : S \times A \times S \to \mathbb{R}$. D is the distribution of starting states. The MDP\R only describes the task environment, of which the trainer's feedback signals are not considered a part.

To give a full specification of interactive shaping, I first establish some notation, which can also be found defined in the notation reference in Section 2.4. The agent experiences a sequence of time steps. The ith such time step (i.e., **e**vent) is referred to as e_i. For each time step e, $e.t_s$ and $e.t_t$ are respectively the clock times of when e starts and terminates. Thus $e_{i+1}.t_s = e_i.t_t$.[3] Likewise $e.s_s \in S$ and $e.s_t \in S$ are respectively the states at the start and end of e. $e.a \in A$ is the action taken by the agent during step e. This action $e.a$ is carried out for the duration of the step e, from $e.t_s$ to $e.t_t$. In addition to these time steps, the agent also experiences feedback

among natural organisms, including humans. "Interactive shaping" differs from the typical usage of "reward shaping" in reinforcement learning literature, which involves adding a supplemental reward to the actual reward [70]. We use "shaping" roughly as it is used in the psychological literature on learning (from where "reward shaping" was also presumably derived). There, "shaping" refers to reinforcing successive approximations of a desired behavior, a teaching strategy that we suspect almost any trainer of an interactively shapable agent will use for complex tasks.

[2] As we approach it, shaping does not require the environment to be Markovian. Regardless, this thesis focuses on problems described as MDPs or MDP\Rs.

[3] Since $e_i.t_t$ is unused by the algorithms in this thesis, I drop the disambiguating subscript on $e_i.t_s$, writing it as $e_i.t$. $e_i.t_t$ is included in the problem definition of interactive shaping for generality.

signals. The ith feedback signal is can be similarly subscripted as h_j to signify the order in which they are experienced. For a feedback signal h, $h.v$ is the value of the human reward signal, and $h.t$ is the clock time at which the reward signal was received by the agent.

I specify interactive shaping below.

The Interactive Shaping Problem While learning in an MDP\R, an agent experiences time steps $E_{hist} = \{e_0, e_1, e_2, ..., e_m\}$. The agent also asynchronously experiences a growing set of feedback signals $H_{hist} = \{h_0, h_1, h_2, ..., h_n\}$ over time.[4] Each feedback signal h_j is received from a *human* trainer who observes the agent and is trying to teach a specific task. The trainer qualitatively understands a task performance metric τ and is trying to maximize the agent's evaluation performance on τ, which is unobservable to the agent.

The instructions and other preparation given to the trainer must only communicate to the trainer that higher reward signals are good, positive, approving, rewarding, or something similar and that lower reward signals are bad, negative, disapproving, punishing, or something similar. The feedback interface need only provide means for the trainer to give at least two different values of feedback at approximately any time.

Given E_{hist} and H_{hist}, how can an agent learn the best possible task policy, $\pi : S \rightarrow A$, as measured by the task performance metric τ?

The interactive shaping problem is nebulously defined with respect to source of the feedback (i.e., the human trainer via the feedback interface). Any change in the trainer's experience—including trainer preparation, the feedback interface, or the type of observation by the trainer—will likely also affect what feedback the trainer

[4] The formulation of the interactive shaping problem allows for any number of reward signals at any time. Thus, at any time the sizes of E_{hist} and H_{hist} are likely to differ greatly, and the relation between i and j for e_i and h_j is not relevant in this thesis.

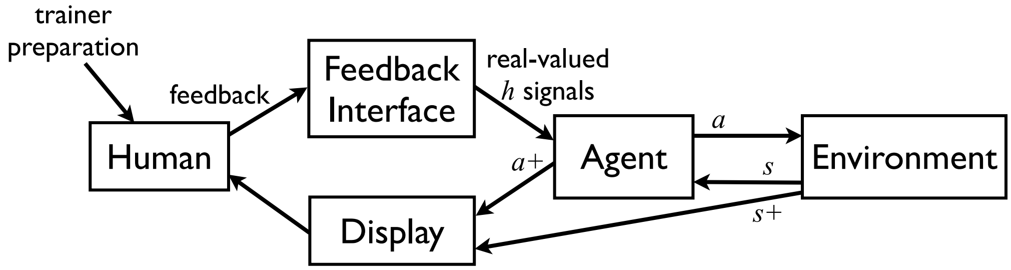


Figure 2.1: Information flow before and during interactive shaping. The symbols $s+$ and $a+$ indicate that more than simply the state and the action might be provided for display purposes. For instance, the agent might communicate its confidence or its planned next action. In this thesis, the display does not receive any such extra information from the agent or environment.

gives. However, we hope and expect that large commonalities exist between various settings; in other words, a good solution to one setting will generally also be a good solution to another.

Further, the problem is underspecified from the agent's perspective. The objective function is unknown to the learning agent, and correct usage of human reward is not addressed. Therefore, since the human reward is the only feedback signal received by the agent, any meaningful solution to the interactive shaping function must map human reward to some objective function. The agent's task performance will be dependent on the quality of this mapping. Underspecification occurs because this mapping is the main open question posed by the problem of interactive shaping, and therefore it cannot be specified as part of the problem.

Figure 2.1 shows information flow before and during interactive shaping. In this thesis, I focus on the specification of the agent, varying the human trainer's preparation, the display, and the feedback interface only by informal optimization. Nonetheless, a formal analysis of possible variations among these components is a valuable direction of inquiry.

In this thesis, I restrict the Interactive Shaping Problem to domains with a predefined task performance metric to allow experimental evaluation. Further, the trainers are assumed to understand a qualitative version of the performance metric. However, interactive shaping will also be helpful when no metric is defined, as for example would likely be the case with an end-user training a service robot.

2.4 Notation introduced in this thesis

This section defines notation that is introduced for this thesis. A few expressions—R_H, $\hat{R}_H$, and $\hat{h}$—are used throughout the thesis, whereas the rest are found only in this chapter and Chapter 3.

- R_H - the human trainer's reward function, which we assume to be of the form $R_H : S \times A \rightarrow \mathbb{R}$ (explained fully in Sections 3.1 and 3.1.1).
- $\hat{R}_H$- a model of R_H, $\hat{R}_H$: $S \times A \rightarrow \mathbb{R}$, also called the "human reward model" or sometimes simply the "human model" (introduced in Section 3.1).
- e - a time step (e.g., an **e**vent), sometimes subscripted with an index—as in e_i—when differentiation between steps is needed for clarity.
  - $e.t_s$ - the clock time that time step e starts.
  - $e.t_t$ - the clock time that time step e terminates.
  - $e.s_s \in S$ - the state of the task at the beginning of step e. When the state at termination is not being considered, $e.s$ or simply s may be used.
  - $e.s_t \in S$ - the state of the task at the end of the step e.
  - $e.a \in A$ - the action taken by the agent during step e. When one arbitrary step is being considered, a may be used as shorthand.
- E_{hist} - a set of time steps experienced by the agent. The set E_{hist} grows with further training experience. $E_{hist} = \{e_0, e_1, e_2, ..., e_m\}$.

- h - a feedback signal received by the agent from an observing human trainer. The signal is denoted with a subscript, as in h_j, when differentiation between feedback signals is helpful. These signals are converted to $\hat{h}$ labels, which are then used to create samples for learning $\hat{R}_H$.
    - $h_j.v \in \mathbb{R}$ - the value of the human reward signal.
    - $h_j.t$ - the clock time the reward signal was received by the agent.
- H_{hist} - a set of feedback signals experienced by the agent over time, typically growing with further training experience. $H_{hist} = \{h_0, h_1, h_2, ..., h_n\}$.
- $\hat{h}$ - the agent's assessment, after accounting for delay in reward, of what reward was intended for specific time step. Section 3.3 explains the process of calculating $\hat{h}$.
- $e.\hat{h}$ - $\hat{h}$ for the time step e.

2.5 Related work: learning from a human

Work on human-teachable agents has taken many forms, almost always with the aim of using more natural modes of communication either to empower people without programming skills to specify desirable behavior or to use the knowledge of such people to increase learning speed for some predefined task. Interactive shaping, in its form as strictly defined in Section 2.3, fits the former goal. In this section, I describe past work on interactive shaping and then on two other forms of teaching: through advice/instruction and by demonstration. In addition to these three categories of teaching modalities, other work has focused on allowing the human to shape the agent's learning environment, facilitating the learning process [85, 108]. Chapter 4 discusses the latter goal, where a task is predefined through a programmed objective function and human interaction facilitates improved learning; related work for learning from human input and a predefined reward function is covered in its Section 4.5.2.

The relative strengths and weaknesses of interactive shaping put it in a unique space that is not currently occupied by any other approach. Furthermore, we believe that many of the approaches I review here are complementary to ours: an ideal learning agent might combine elements from several of them. Indeed, a recent Wizard-of-Oz study (where the learning agent is secretly a human) examined teaching patterns when humans were given several different teaching methods to use [42, 110]. In this study, a prevalent pattern was to teach by demonstration, test the learned skills, and then give reward-style feedback; this pattern is an intuitive strategy for interspersing different teaching modalities.

2.5.1 Interactive shaping

This thesis is not the first research to address the interactive shaping problem. In Figure 2.2, a comparison of various past approaches is presented, focusing on those methods that allow both positive and negative rewards.

When the trainer can only give positive rewards, this method is sometimes called *clicker training*, which comes from a form of animal training in which an audible clicking device is previously associated with reward and then used as a reinforcement signal itself to train the animal. Previous work on clicker training has involved teaching tricks to entertainment agents. Kaplan et al. [43] and Blumberg et al. [8] implemented clicker training on robotic and simulated dogs, respectively. Blumberg et al.'s system is especially interesting, allowing the dog to learn multi-action sequences and associate them with verbal cues. Though important to research on interactive shaping in that they are novel techniques of teaching pose sequences to their respective platforms, neither is evaluated using an explicit performance metric and the methods are not framed to be generally applicable to sequential decision-making tasks.

The first work we have found to use human reward (both implicit and explicit in this case) to train an agent is by Isbell et al. [37, 36]. In this research, a social software agent named Cobot learns human preferences in LambdaMOO, an online, text-based virtual world where people interact. The agent uses reinforcement learning "to proactively take action in this complex social environment and adapt his behavior from multiple sources of human reward."

Thomaz and Breazeal [105, 107] interfaced a human trainer with a table-based Q-learning agent in a virtual kitchen environment. Their agent seeks to maximize its discounted total reward, which for any time step is the sum of human and MDP reward. This approach, adding a supplementary reward function to the MDP's reward function, is a form of what is sometimes called "reward shaping" in the reinforcement learning literature [70].[5] In this work, they also show improved performance when the trainer can additionally give action guidance. Another study extended their work to a robotic object-sorting domain [88], using only human reward (no MDP reward).

Tenorio et al. [101] expand on Thomaz and Breazeal's algorithms, creating an algorithm that learns from explicit demonstration as well as human and MDP reward. As in Thomaz and Breazeal's work, the two reward signals are added together and seen as a single signal for the agent. For Tenorio et al.'s algorithm, all human input comes from verbal commands or feedback.

More recently, Pilarski et al. used a continuous-action actor-critic RL algorithm to learn a myolectric control policy for a simulated robotic prosthetic arm, using human reward signals as feedback [76]. In their experiments, human subjects taught a continuing task of matching the robotic arm to various target joint angles that are repeated in order throughout the task. The policy inputs were robotic

[5] Note that this use of the word shaping differs from its main usage in this article.

Earliest publication	**Number of tested domains**	**Reward interface**	**Interface to reward value mapping**	**Human reward model repr.**	**Models reward delay?**	**Can learn from more than one reward per step?**	**Other human inputs**	**MDP reward?**	**Episodic or continuing tasks tested**	**Effective Discount Factor**	**Value function repr.**
Isbell et al., AGENTS 2001	1	Typed verbs; explicit rewards ("reward", "punish") and implicit rewards (e.g., "hug" and "spank")	Values not reported; explicit reward values are larger than implicit ones.	Not modeled	Implicitly through RL bootstrapping	Implicitly by adding multiple rewards	No	No	Continuing	γ = 0.7	Linear model (with socially relevant features)
Thomaz and Breazeal, AAAI 2006	1	Dragging mouse	Drag direction and length maps to value within [-1, 1]	Not modeled	Implicitly through RL bootstrapping	Implicitly by adding multiple rewards	Action guidance in some versions	Yes	Episodic	γ = 0.75	Tabular
Knox and Stone, ICDL 2008 (**TAMER**)	6	Push-buttons	1 and -1	Linear models and kNN	Yes (explicitly)	Yes	Manipulation of state in interactive robotic task; otherwise no	In some versions	Episodic and continuing	γ = 0	N/A, value and reward functions are equivalent
Tenorio et al., IBERAMIA 2010	2	Verbally delivered words (in Spanish)	"terrible" = −50, "bad" = −10, "good" = 10, and "excellent" = 50	Not modeled	Implicitly through RL bootstrapping	No	Demonstrations	Yes	Episodic and continuing	γ = 0.9	Tabular
Pilarski et al., ICORR 2011	1	Push-buttons	0.5 and - 0.5	Not modeled	Implicitly through RL bootstrapping	N/A; time steps are merely 5 ms.	Myoelectic signals converted to state values	No	Continuing	γ = 0.99 (with 5 ms time steps, reward a second ahead is discounted by a factor of 0.134)	Linear model (with tile coding features)
Suay and Chernova, Ro-Man 2011	1	Dragging mouse	Drag direction and length maps to value within [-1, 1]	Not modeled	Implicitly through RL bootstrapping	No	Action guidance in some versions	No	Continuing	γ = 0.75	Tabular
Knox, **Chapter 6** in this thesis	1	Push-buttons	1 and -1	Linear model	Yes (explicitly)	Yes	No	No	Episodic and continuing	γ is 0, 0.7, 0.9, 0.99, or 1.0	Tabular

Figure 2.2: A comparison of various solutions to interactive shaping or similar problems. Citations follow for each row's publication, from top to bottom: [37, 105, 50, 101, 76, 88]

joint angles and measures of electrical activity at four different arm muscles, and the actions consisted of two joint velocities.

From this related research on interactive shaping, I note several patterns that highlight novel contributions of this thesis or hint at insights in later chapters.

1. Only the algorithms in this thesis are implemented and tested on more than 2 task domains. See Section 3.4 for a list of task domains in this thesis.

2. The positive and negative reward values that the human feedback interface maps to are highly balanced. In other words, no one has performed experiments where, say, the only positive reward signal maps to a $+1$ and the only negative reward signal maps to a -10. This pattern is related to observations of bias towards positive reward, which are especially important in Chapter 6.

3. Besides the algorithms developed in this thesis, no interactive shaping algorithms explicitly model human reward.

4. In all of the related work that involves learning from positively and negatively valued reward signals, the authors take a reinforcement learning approach, an intuitive choice given that the tasks are sequential decision-making problems and the humans are giving evaluative feedback. Additionally, every algorithm uses curiously low discount factors. In reinforcement learning, discount factors are almost always at or near 1. A discount factor of 0.9 is somewhat low, and 0.7 or 0.75 is quite low. Even Pilarski et al., with $\gamma = 0.99$, is effectively low; time steps are 5 milliseconds, so reward one second in the future retains only approximately 13.4% of its immediate value. These choices of low discount factors foreshadow my focus on discounting throughout this thesis, especially in our chief approach to interactive shaping—the TAMER framework (Chapter 3)—and our investigation of different discounting in Chapter 6.

5. Outside of this thesis, no algorithms explicitly model the delay in human reward. Instead, they account for delay implicitly through the temporal-difference bootstrapping of reinforcement learning. But given their fairly myopic discounting, delay is not accurately modeled; specifically, there is a minimal response period that should receive no credit but instead receives large credit (see Sections 3.3.1 and 3.3.2 for relevant discussion).

6. Various methods differ in their ability to handle more than one reward per time step. Note that the credit assignment method described in Section 3.3 allows our algorithms to attribute an infinite number of reward values to a time step (within the range bounded by the number of times a trainer can push a button per time step), despite the mapping of reward keys to $+1$ and -1, giving a richer feedback signal than the interface suggests.

7. Some algorithms incorporate demonstration and action guidance from the human. Others allow the human to affect the task's state. A few algorithms also incorporate MDP reward. In this thesis, I consider learning only from human reward (Chapters 3, 5, and 6) and learning from both human and MDP reward (Chapter 4).

2.5.2 Learning from advice or instruction

In the context of agent learning, instruction and advice can be thought of as providing a desired subset of actions when a certain condition is true. For example, a soccer coach might give advise, "If you are near one side of your own goal and a potential pass target is on the other side, do not pass to that target." Following their colloquial meanings, instruction and advice differ in how the agent uses them; instruction commands an agent, whereas advice merely suggests action. Since advice is merely suggestive, it assumes the presence of another evaluative signal that provides ground-truth judgement of behavior, such as MDP reward (as we do in

Chapter 4).

Giving advice or instruction via natural language is a promising way for non-technical humans to teach agents. Natural language advice was first demonstrated by Kuhlmann et al. [57], who created a domain-specific natural language interface for giving advice to a reinforcement learner.

The informational richness of advice is clearly powerful. However, general natural language recognition is unsolved, so many current advice-taking systems [64, 68] require that the human encode her advice into a scripting or programming language, making it inaccessible to non-technical users. The natural language component of Kuhlmann et al. [57] required manually labeled training samples.

Seeking to learn policy-related language more naturally, Goldwasser and Roth created a system that translates natural instruction to actionable game rules for Freecell [31]. Actions were labeled as legal or illegal by a game API, and these labels were used to improve the translation model. The model's final translation accuracy ranged between approximately 60 and 90% on various subtopics of the rules, and in all cases the initial model was improved upon considerably. Such a correct/incorrect labeling is quite similar to interactive shaping and could potentially be given by a non-technical user.

In work by Chen and Mooney [17], an algorithm learns how to follow natural-language instructions in a system of corridors with landmarks and other location-distinguishing characteristics. Specifically, human users controlled an agent through the environment to fulfill specific instructions within a corpus, creating demonstrations of how to follow those instructions. Using the set of demonstrations for known instructions, their algorithm creates a semantic parser that maps novel instructions to a planned navigation trajectory.

There will likely be times when the trainer knows that the agent has performed well or poorly, but cannot articulate exactly why. In these cases, positive or negative feedback will be easier to give than advice.

2.5.3 Learning from demonstration

Another way for a human to teach an agent is to demonstrate a task via remote-control or his own body, while the agent records state-action pairs from which it learns a general policy for the task [4]. Using the meaning of "instruction" from Section 2.5.2, demonstration can technically be seen as a special case of instruction, where the condition is always the current or next state and the subset of actions contains only the single action that the human selects. Successes of learning from demonstration include the task domains of autonomous driving [77], multi-robot coordination [19], robot soccer [32], and helicopter flight [2].

In Apprenticeship Learning [2], a type of learning from demonstration, the algorithm begins with an MDP$\backslash R$ (as does the TAMER framework; see Section 2.3). The algorithm learns a reward function R, using inverse reinforcement learning, that could justify a human's behavior during a period of control, and the agent follows the policy that maximizes expected return under that reward function. Apprenticeship learning differs from interactive shaping by the input the learner gets from humans: reward for interactive shaping and demonstrations for apprenticeship learning.

Demonstrations and action interventions (changing or influencing autonomously chosen action) have also been used to learn specific information rather than entire policies. For example, Ugur et al. had human teachers apply force to change the default trajectory of a robotic arm. When the robot's hand made contact with an object, it would close its grip and attempt to lift the object. If the lift was successful, the first-contact points from the human-influenced trajectory were later used as positive samples in learning a classifier for graspable locations of various objects [111], which can be used to plan a grasping trajectory.

Learning from demonstration has some advantages over interactive shaping.

First, despite a reward/punish training interface generally being simpler than a control interface, many people are already familiar with control interfaces through activities such as driving and playing video games. An interactive shaping interface will likely be novel, adding to the amount of preparation a trainer needs. But since it is simpler, an interactive shaping interface may be easier to use than a control interface after reasonable experience. Additionally, learning from demonstration will often lead to faster learning, since the correct action can be directly communicated by the demonstrator; during interactive shaping, the correct action can be encouraged by reward if it is taken, but if it is not, punishment will only communicate to take any action *other* than the one actually taken.

There are some tasks that are too difficult for a human trainer, especially when demonstration requires the trainer to control the agent. The agent might have more actuators than can be put in a simple interface (as is common for robots), the task may require the human to be an expert before being able to control the agent (e.g., helicopter piloting in simulation), or good task behavior might entail high-frequency actions or response times that a human cannot provide. In these cases, a demonstration is infeasible. But if the human can judge the quality of the agent's behavior, then he or she should be able to provide feedback via TAMER regardless of the task's difficulty. Further, demonstration is conventionally delivered without any means for the trainer to see what the agent has learned. If the policy learned by the agent produces problematic behavior, the demonstrator may never know and thus be unable to adapt his or her demonstrations to the learner. On the other hand, agents always express their learned behavior during interactive shaping, and the trainer can adjust feedback to address issues with learned behavior. Lastly, an interactively shaped agent could learn a policy that outperforms the trainer's intended policy on the task performance metric that the trainer is attempting to maximize, an outcome that only occurs through randomness or bias when teaching only from demonstration. Chapter 6 explores this possibility through planning to maximize long-term reward, which does not occur with TAMER's myopic action selection. This potential ability to outperform the trainer's desired policy suggests a powerful scenario combining demonstration and reward. A trainer could give a rudimentary demonstration while giving reward at critical points, and the agent could plan a more effective path to accrue predicted human reward; the path might greatly improve upon the low-quality demonstration, which is still necessary to get to the critical points that received feedback.

Chapter 3

TAMER

This chapter presents a solution to the Interactive Shaping Problem specified in Chapter 2, the problem of learning tasks from human-generated reward. [1] *This solution, which we dub the* TAMER *framework, is built upon and investigated further in Chapters 4 and 5. Chapter 6 examines alternate algorithms that generalize from* TAMER.

TAMER *differs from past approaches to learning from human reward in three key ways: (1)* TAMER *addresses delays in human evaluation through credit assignment, (2)* TAMER *learns a model of human reward ($\hat{R}_H$), and (3) at each time step,* TAMER *chooses the action that is predicted to directly elicit the maximum reward ($argmax_a \hat{R}_H(s,a)$), eschewing consideration of the action's effect on future state.*

TAMER *is fully implemented. This chapter reports on three user studies in two domains in simulation and then describes an implementation on a robot that is informally evaluated.* TAMER *agents, when compared to agents learning from a predefined reward function (as part of each task's Markov decision process specification), reach qualitatively good performance more quickly. Additionally, note that* TAMER *differs fundamentally from past work by its discounting of future human*

[1] This chapter encompasses and extends previously published research by the author [50, 47, 49].

reward; Chapter 6 examines the effect of different discounting assumptions.

Learning agents have the potential to impact our lives greatly, learning control strategies for applications from autonomous driving to neural prosthetics to completing time-consuming but mundane tasks such as household chores. As I argue in Chapter 1, teaching by human-generated reward addresses two areas of agent learning algorithms that must be improved to make them widely applicable to real-world tasks: learning speed and an ability to adapt to end-users' needs without further programming. In this chapter, I introduce the TAMER framework for learning from human reward, addressing the problem of interactive shaping that was specified in Chapter 2.3.

In Section 3.1, I introduce the TAMER framework, giving intuition for its high-level algorithmic choices; unlike past work, TAMER uniquely treats human reward as different from MDP reward. Section 3.2 describes a simple TAMER algorithm for domains with low action frequencies, for which credit assignment is trivial, and Section 3.3 details our approach for assigning credit from human reward in domains with more frequent actions. In Section 3.4, I describe the instantiated components of TAMER for three tasks, Tetris (more challenging but with infrequent actions), mountain car (less challenging but with frequent actions), and interactive robotic navigation. I report and discuss our experimental results on these tasks in Section 3.4 and review related work in Section 2.5.

The experiments in the first two domains compare TAMER agents to tabula rasa reinforcement learning (RL) algorithms. The TAMER agents learn good behavior much more quickly, and the RL algorithms meet or surpass TAMER agents' performances when given enough learning samples. This complementary role of the TAMER and RL algorithms is explored in Chapter 4.

3.1 The TAMER framework for interactive shaping

Before I formally present TAMER algorithms in Sections 3.2 and 3.3.6, I first motivate and describe the TAMER framework at a high level.

Consider the Interactive Shaping Problem. The agent's role is by nature somewhat underspecified: it must learn from human reward such that it maximizes task performance—which it cannot measure. We hypothesize that the agent best fills its role by learning a model of the trainer's reward and selecting actions that the model predicts will directly result in the most reward. More specifically, TAMER breaks the process of learning behaviors from live human reward into three modules (Figure 3.1): (1) credit assignment, where delayed human reward is associated appropriately with recent events; (2) regression on experienced events and their consequential credited reward to create a predictive model for future reward; and (3) action selection using the model of human reward. Setting aside the problem of credit assignment until Section 3.3, I discuss TAMER's approach to learning a model of human reward and action selection below.

A TAMER agent seeks to learn the human trainer's reward function, which we assume to be of the form $R_H : S \times A \to \mathbb{R}$ (more on this assumption in Section 3.1.1). Presented with a state s, the agent consults its learned model $\hat{R}_H$ and, if choosing greedily, takes the action a that maximizes $\hat{R}_H(s,\ a)$.[2] Figure 3.1 shows interaction between a human, the environment, and a TAMER agent within an MDP\R. Since the agent seeks only to maximize human reward, the optimal policy is defined solely by the trainer, who could choose to train the agent to perform any behavior that its model can represent. When the agent's performance is evaluated using an objective metric, we expect its performance to be limited by the information provided by the teacher.

[2] When describing arbitrary steps that are input to R_H or $\hat{R}_H$, I sometimes use (s, a) or (e) as shorthand for $(e_i.s, e_i.a)$.

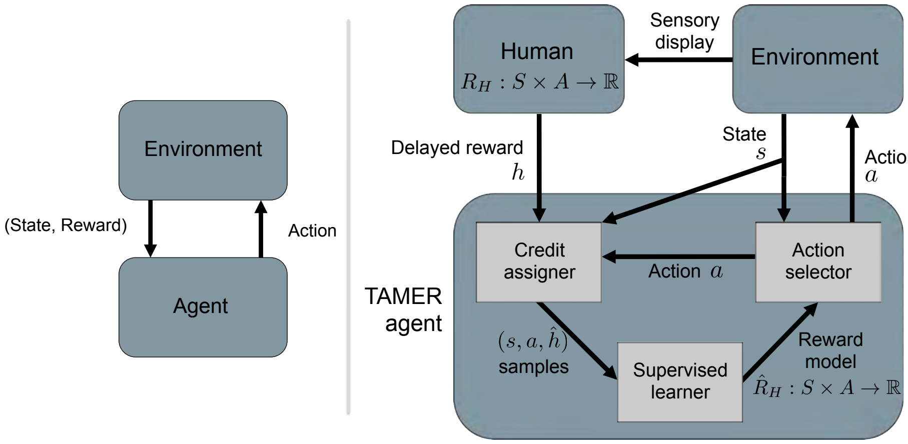


Figure 3.1: On the left, the traditional scheme of interaction between agent and environment in a Markov Decision Process. Many different instantiations of the agent exist in the literature, including algorithms such as Q Learning [112], SARSA(λ) [90], and LSPI [59]. On the right, the framework for Training an Agent Manually via Evaluative Reinforcement (TAMER).

3.1.1 Modeling human reward

TAMER learns a function $\hat{R}_H$ that approximates the expectation of reward given by a human trainer for each possible state-action pair: $R_H : S \times A \rightarrow \mathbb{R}$. The function R_H, the true expectation of reward, is called the "human reward function" throughout this chapter. This specific formulation of R_H results from four *assumptions* that make $\hat{R}_H$ more tractable to learn and more practical for action selection:

1. *R_H does not take non-task information as input.* The state and action inputs to R_H are specific to the task being performed. Other observations that might improve the predictive quality of $\hat{R}_H$, such as the history of past rewards (H_{hist}) and indications of the trainer's attentiveness (e.g., gaze direction), are not used. This assumption, though, allows $\hat{R}_H$ to specify behavior in any situation—especially without the trainer present—because $\hat{R}_H$ is independent of information beyond the bounds of the task.

2. *Human reward targets state and action at a single point in time.* In violation of this assumption, human reward could be influenced by any subset of the agent's state-action history (E_{hist}). This assumption simplifies TAMER's learning problem greatly while still allowing enough expressiveness to learn well. Our credit assignment method, described in Section 3.3, partially accounts for the flaws of this assumption by spreading the credit for a reward among multiple state-action pairs that are considered possible targets of the reward. I briefly discuss the violation of this assumption in Section 3.3.5. In Section 7.1's list of potential future projects, I consider approaches that avoid this assumption under the item "Non-Markovian models of human reward".

3. *The specific distribution of reward for a state-action pair does not contain evaluative information that is not already captured by the expectation of reward for the pair.* Whether human reward is delivered and its value when delivered appears to be highly stochastic. In reinforcement learning, a stochastic reward function can be replaced by a reward function that outputs the expectation of the original without changing the values of state-action pairs for any specific policy. In this spirit, TAMER algorithms learn a deterministic model of human reward that approximates the expectation of reward. Some intuitively appealing extensions to the algorithms in this thesis would require relaxing this assumption. For instance, the agent could use its confidence in its assessment of the expectation of reward to either influence when it explores or to communicate a need for further training, creating an paradigm related to active learning (similar to some past work on learning from demonstration [18]).

4. *Human reward is not dependent on transitions between states.* Trainers likely give reward based on both actions and their consequences, which would be captured by an alternate human reward function $R_H : S \times A \times S \rightarrow \mathbb{R}$, where the second state input results from the action taken during the first

state input. As for other assumptions, TAMER ignores the resultant state to simplify the input space. This assumption also avoids requiring the agent to predict future state while choosing an action, a significant restriction. Like the second assumption, the relaxation of this assumption is addressed in the item "Non-Markovian models of human reward", found in Section 7.1's list of potential future projects.

Removing any of these assumptions results in a richer representation of human reward. But with any larger input space the number of learning samples needed to learn a good policy will likely grow. Further, removing the first or fourth assumptions would reduce the applicability of $\hat{R}_H$. We leave for future work an investigation of the correct balance of representational richness and simplicity.

Learning $\hat{R}_H$ is a supervised learning problem, given a credit assignment method that appropriately uses delayed reward to create labels for state-action pairs (see Section 3.3). Intuition and some evidence [36] suggest that a human trainer's reward function R_H (as we have defined it) is a moving target for learning. Intuitively, it seems likely that a human trainer will raise his or her standards as the agent's policy improves, expecting more to get the same reward. Though we have not rigorously studied the claim that a human's reward function is a moving target, our experiments do indicate that both the ratio of positive to negative reward and the frequency of feedback change over time (Section 3.4.3); however, the distribution of state-action pairs also differs over time, so our experiments merely suggest that R_H changes. Considering this conjecture that R_H is a moving target function and that the TAMER agent should be able to exploit new data as it becomes available, the regression algorithm should both be able to handle changes in the target function and allow quick, online updates. Quickly incorporating new feedback increases the agent's responsiveness, which we expect to improve the human trainer's level of engagement and to increase the relevance of feedback by presenting the most current behavior for evaluation by the trainer. Additionally, we believe that the algorithm should generalize to a large proportion of, and possibly all of, the unseen samples (i.e., across states and possibly actions). Examples of such algorithms include k-nearest neighbors and incremental gradient descent on a linear model over generalizing features like Gaussian radial basis functions. (Both are used at various points in our experiments).

3.1.2 Action selection

Autonomous learning agents, as I speak of them, receive feedback from a predefined reward function within an MDP (i.e., reinforcement learning agents [90]). For those autonomous agents that modify their behavior by learning an estimate of each state-action pair's value, the principal challenge is to assign credit from received reward to the entire history of past state-action pairs. In domains where the only discriminating reward is at the end of the task, assigning this credit can be particularly difficult. A key insight of the TAMER framework is that the problem of credit assignment inherent in reinforcement learning is not necessarily present with an attentive human trainer. We hypothesize and indeed observe that the trainer evaluates an action or short sequence of actions with a model of the long-term consequences, making the evaluation a full judgement on the quality of the behavior—in direct contrast with evaluation from a traditional reward function. Further, the trainer can deliver his or her feedback within a small temporal window after the behavior. Assigning credit within that window presents a challenge in itself, which I discuss in Section 3.3, but it is much easier than assigning credit for an arbitrarily delayed reward signal as in reinforcement learning. Assuming that credit is properly assigned within the temporal window, a trainer can directly label behavior. As I have described, modeling the trainer's reward function R_H is a supervised learning problem. Action selection follows trivially by exploiting $\hat{R}_H$, the outputs of which predict the trainer's full judgement on a state and action. *Therefore,* TAMER*'s process of learning a policy from human reward reduces an apparent reinforcement learning problem to a supervised learning problem with non-trivial but easier credit assignment.*

More specifically, to choose actions within some state s, a TAMER agent directly exploits predictions of expected human reward from the learned model $\hat{R}_H$. When acting greedily, a TAMER agent chooses the action $argmax_a[\hat{R}_H(s,a)]$.

The TAMER framework is agnostic to *exploration*, leaving action selection as a module to be specified by the agent designer. However, we have found that greedily choosing the action that is expected to receive the highest human reward provides sufficient exploration for many tasks, including all of those described in this thesis. This counterintuitive finding is easily justified. In a given state, a good trainer can negatively reinforce an undesired action, eventually dropping its expected reward below that of another action, resulting in a new (exploratory) action choice. A non-greedy action selection method may be needed if there are actions of which the trainer is not aware or does not know the value. Otherwise, the trainer should be able to guide the TAMER agent's exploration sufficiently under greedy action selection.

3.1.3 Comparison with reinforcement learning

Before formally presenting the first TAMER algorithm, I first compare the framework to reinforcement learning, focusing on their respective assumptions. The TAMER framework for shaping agents shares much common ground with RL, but there are some key differences that are important to understand to fully appreciate TAMER.

In reinforcement learning, agents seek to maximize return, which is a discounted sum of all future reward. In contrast, a TAMER agent does not seek to maximize a discounted sum of all future *human* reward. Instead, it attempts to directly maximize the reward attributable to the chosen state-action pair without consideration of the pair's impact on future states.[3] A TAMER agent does this because the trainer's reward signal is assumed to be a direct label on the quality of recent state-action pairs.

Correspondingly, the human's reward function R_H is not an exact replacement for a reward function R within an MDP. Indeed, TAMER's interpretation of a human's R_H is closer to that of an action value function than of an MDP reward function. Although it may be possible for a reinforcement learning algorithm to use R_H in lieu of a reward function, naively doing so ignores fundamental differences between MDP reward and human reward. In Chapter 6, we explore the relationship between human and MDP reward, validating the necessity of TAMER's myopic action selection in a large class of tasks; in that chapter, we subsequently identify a task manipulation that allows an algorithm that does consider long-term reward — a generalization of TAMER— to learn effectively.

Additionally, in this chapter we use MDP\Rs both to investigate how to learn in the absence of a reward function and because the reward function within an MDP dictates the optimal policy; without R, the human's reward function is the sole determinant of good and bad behavior, fitting the description of interactive shaping.

3.2 TAMER algorithm for low action frequencies

Algorithm 4 shows the TAMER algorithm for tasks in which the agent acts infrequently enough that the human can unambiguously give reward to a single state-action pair. This algorithm uses a simple form of credit assignment; human reward accrued during one step's action is credited to the previous step's state action pair.

[3] I sometimes refer to the objective of a TAMER agent as maximizing "immediate reward". I use this phrase as shorthand for the longer, more precise description in the previous sentence.

Algorithm 4 The TAMER algorithm for infrequent actions

```
Main thread
 1: $i \leftarrow -1$
 2: while $true$ do
 3:     $i \leftarrow i + 1$
 4:     if $trainer\ is\ engaged$ and $i \geq 2$ then
 5:         $updateModel(\hat{R}_H,\ (e_{i-2}.s, e_{i-2}.a, \hat{h}))$        // Label second-to-last state-action pair
 6:     end if
 7:     $\hat{h} \leftarrow 0$
 8:     $e_i.s \leftarrow getState()$
 9:     $e_i.a \leftarrow argmax_a \hat{R}_H(e_i.s, a)$
10:     $takeAction(e_i.a)$
11:     wait for next time step
12: end while
Human interface thread
 1: while $true$ do
 2:     if $new\ reward\ h\ with\ value\ h.v\ is\ received$ then
 3:         $\hat{h} \leftarrow \hat{h} + h.v$
 4:     end if
 5: end while
```

In Section 3.3, we explore more general forms of credit assignment, one of which later extends this algorithm to create the full TAMER algorithm (Algorithm 5).

This algorithm uses two threads. The main thread loops once per time step and the human interface thread loops constantly (in practice, usually with a small delay between loop iterations) to receive human reward as it is given.

After the agent initiates a step counter i, it enters the loop of the main thread (line 2), which iterates once per time step. At the start of each loop, the agent increments i and then updates $\hat{R}_H$ if the trainer is engaged (lines 4–6). The advantage of considering trainer engagement is that the agent can act interminably with a static policy when the trainer is disengaged. Determination of trainer engagement is an interface design issue. In our first implementation of TAMER, if the human does not give reward (i.e., $\hat{h} = 0$), we conservatively assume that the trainer is not currently training, and $\hat{R}_H$ is consequently not updated. In our most current implementation, the trainer has a button to flip a boolean flag that determines whether a training session is occurring. Both work well, but we prefer the latter approach, which allows a trainer to give zero feedback meaningfully. These two implementations are discussed more extensively in Section 3.3.3. If the model update does occur, the agent creates a supervised learning sample with features from s and a and label $\hat{h}$ to update $\hat{R}_H$ (line 5). Throughout this chapter, $\hat{h}$ denotes the reward label for a state-action pair; $\hat{h}$ is TAMER's estimate of the intended reward for the state-action pair, as inferred from raw rewards, each of which are denoted by h, with value $h.v$ and delivery time $h.t$ (see the definition of this notation and the definition of interactive shaping in Chapter 2 for more detail).

The updateModel() function's second argument is a sample for supervised learning, where the features are $e_{i-2}.s$ and $e_{i-2}.a$ and the label is $\hat{h}$. e_{i-2} is labeled with $\hat{h}$ under the assumption that the action $e_{i-2}.a$ is not fully observed until e_{i-2} terminates and e_{i-1} starts, and the trainer is then able to give reward for the duration of e_{i-1}, making the label for e_{i-2} known at the start of e_i. A trainer interacting with Algorithm 4, as in the Tetris experiments described in Section 3.4, is instructed to wait until an action is complete before giving feedback. Some trainers have difficulty following this instruction; in light of our experiences during experimentation, we consequently recommend using credit assignment as described in Chapter 3.3, even for slow domains. Nonetheless, this simpler algorithm is effective enough to create positive results, and it is useful for explicative purposes and to describe the agents in the Tetris experiments.

The subroutine updateModel() is specific to the supervised learning algorithm used in a particular instantiation of TAMER. For example, gradient descent would adjust the parameters of $\hat{R}_H$, whereas k-nearest neighbors would simply add the new sample to memory. Note that if features are extracted from state and action, this algorithm hides the extraction step within the function $\hat{R}_H$ and possibly within updateModel().

After the potential model update, $\hat{h}$ is set to 0 (line 7), and the agent obtains the new state description, assigning its value to step experience e_i (line 8). The agent then greedily chooses the action that, according to the human reward model $\hat{R}_H$, yields the largest predicted reward (line 9). Although we have found that greedy action selection is sufficient in our work thus far, more sophisticated action selection methods may be needed in some situations and would also fit well within the TAMER framework. Once the action is selected, the agent executes the action and then waits for the next time step (lines 10 and 11).

The human interface thread loops constantly, checking for input from the trainer. If new input h is observed, its value $h.v$ is added to the value of $\hat{h}$ (thus allowing for any single action to receive multiple feedback instances).

3.3 TAMER algorithm for high action frequencies

In some task domains, the frequency of time steps is too high for human trainers to respond to specific state-action pairs before the next pair occurs. In simulation, it is generally possible to lower the frequency of time steps; but doing so may change the character of the task, and in physical domains (e.g. helicopter flight) it is likely impossible. Although we have not studied exactly where the frequency threshold is, our experience and previous studies of human response times (Figure 2) [35] have made it clear that having several seconds between time steps is enough for specific labeling and, conversely, having several time steps per second is too frequent for specific labeling. However, as mentioned in the previous section (3.2), credit assignment may be desirable even for low-frequency domains.

For faster domains, credit from the human's reward must be appropriately distributed across some subset of the previous time steps. In our approach, each human reward signal is shared among recent state-action pairs. In the rest of this section, I develop and analyze TAMER's credit assignment method and then give the general TAMER algorithm, which includes credit assignment. Section 3.3.1 introduces the specific credit assignment problem and some assumptions we make in specifying it. In Section 3.3.2, I describe how to calculate the probability that a single reward signal is targeting a single time step (i.e., an event with a state-action pair). Section 3.3.3 explains how to aggregate all rewards to provide a label for a single time step. Section 3.3.4 compares this labeling method to a prior method used in some of our experiments, and Section 3.3.5 discusses the labeling method's strengths and limitations. Lastly, in Section 3.3.6, I introduce the full TAMER algorithm that uses this credit assignment technique, generalizing Algorithm 4. The description here of our credit assignment method relies heavily on the notation defined in Chapter 2.4.

3.3.1 Feedback delay

To handle delayed feedback, we first attempt to understand the extent of delay. Figure 3.2 shows histograms and fitted probability density functions for the delay in human subjects' responses in visual searching tasks of different complexity [35]. Inspired in part by these results, we define a probability density function f_{delay} that estimates the probability of the trainer's feedback delay. Thus, for some point in time that the trainer targets with feedback, this function provides the probability that the feedback occurs within any specific time interval. Feedback on a point is considered feedback on the state-action pair $(e.s, e.a)$ for the step e within which the point falls. Put more simply, in this *forward view* the question is, "Given a known target of feedback, when will it be rewarded?" However, in practice, a learner has the *backward view*, asking, "Given known feedback, when was the point in time that the feedback targets?" Since different points within the same event e equivalently connect to it, we can more generally ask, "Given known feedback, which event e does the feedback target?" These two perspectives are illustrated in Figure 3.3, where delay is relative to the known event. Given the nature of our learning task, we take the backward view and define f_{delay} with support over negative times, considering the time of the feedback in question to be 0.

Stepping forward from the backward-view question, we define a **problem statement** of credit assignment, which we address in this section: *If human reward signals are received with probabilistic, unobserved delay from the point in time they target, how can* $\hat{R}_H$ *be learned such that it approximates* R_H*, the expectation of reward for each state-action pair?* In other words, how should the algorithm create labels for experienced state-action pairs, assuming an unbiased estimator will train from the generated samples? The rest of Section 3.3 addresses this question. The algorithm I describe for labeling time steps, *delay-weighted, aggregate reward*, gives an approximate solution to the problem statement above.

3.3.2 Calculating probability that a human reward signal targets a time step

Delay-weighted, aggregate reward is built upon calculations of the probability that a human reward signal targets a specific time step. I describe this calculation here and then explain its use in the full calculation of delay-weighted, aggregate reward in Section 3.3.3. In addition to the notation established in Section 2.3, we define the boolean function $targets : H_{hist} \times E_{hist} \rightarrow \{0, 1\}$ to be an unknown (and generally unknowable) function that indicates whether a human reward signal targets a time step.[4]

Under this notation, the probability that a reward h targets a time step e is

[4] The credit assignment method in this section does not assume that each state-action pair results in exactly one human reward signal. Specifically, note that $i = j$ for e_i and h_j does not imply $targets(h_j, e_i)$.

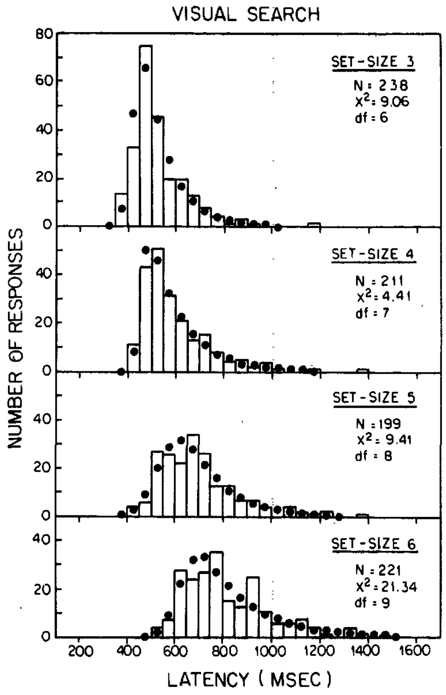


Figure 3.2: Results from a study of the distribution of human response times for visual searches of different levels of complexity [35]. Subjects pushed a button upon identifying an n-sized visual pattern, making this task similar to that of a human trainer—watching behavior and evaluating it. Graphs show histograms of response times and a probability density function (constrained to the sum of a normal random variable and an exponential random variable) fit to the histograms. Figure reprinted with the author's permission.

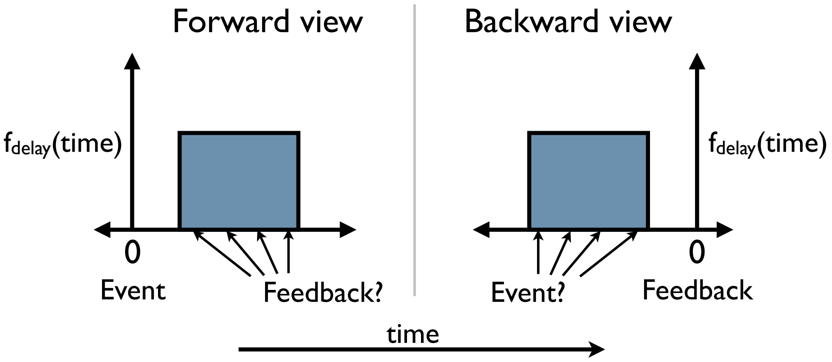


Figure 3.3: These two plots show the forward and backward views of the probability of delay in human reward. Our credit-assignment problem fits the backward view.

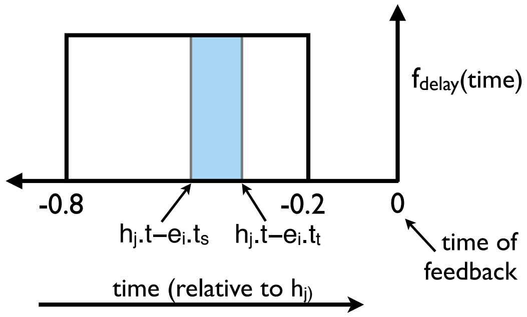


Figure 3.4: This plot illustrates the calculation of the probability that a specific reward h targets a single time step e, using the distribution Uniform(-0.8 seconds, -0.2 seconds) for probability density function f_{delay}. The probability is calculated as $P(targets(h, e)) = \int_{h.t-e.t_s}^{h.t-e.t_t} f_{delay}(x)dx$.

$$P(targets(h, e)) = \int_{e.t_s-h.t}^{e.t_t-h.t} f_{delay}(x)dx.$$

This calculation is illustrated in Figure 3.4 for a uniform distribution as f_{delay}. The integral bounds are the time lapsed from the start and end of the time step e to the reception of feedback h. Since feedback must come after its target in time, these time-lapse values are negative.

Figure 3.2 indicates that human trainers cannot respond in less than approximately 0.2 seconds. We therefore generally recommend specifying f_{delay} such that it does not have support after -0.2 seconds.

Differences in individual response times, the complexity of the task, and other factors will likely affect the actual distribution of response delays in any one training episode. Therefore calibrating f_{delay} to a specific trainer and task may improve the accuracy of $\hat{R}_H$, an approach we do not explore.

3.3.3 Delay-weighted, aggregate reward

Section 3.3.2 considered how, given a *single* human reward signal, to calculate credit for a specific candidate time step. In this section, I broaden to describe how to create a label for each sample $(e.s, e.a)$ that will lead an unbiased estimator to approximate the expectation of human reward, R_H, for a general occurrence of (s, a), assuming infinite samples and that the expectation does not change over time. The label created by the described technique (and the technique itself) is termed *delay-weighted, aggregate reward*; this label is denoted $e.\hat{h}$. As I will discuss at the end of this section, though this label will not learn the expectation $R_H(s, a)$ exactly, it has desirable qualitative characteristics and can learn effective policies in practice. Delay-weighted, aggregate reward, $e.\hat{h}$, is defined as the sum of rewards weighted by each reward's probability of targeting e:

$$e.\hat{h} = \sum^{h \in H_{hist}} h.v \;\; P(targets(h, e)) = \sum^{h \in H_{hist}} h.v \int_{e.t_s - h.t}^{e.t_t - h.t} f_{delay}(x)dx \qquad (3.1)$$

In principle, f_{delay} may have a long tail; in other words, when a human reward signal is observed, there's some small chance that it was meant to target an event that occurred in the distant past. However, in practice, we expect that most of the probability mass of f_{delay} will be within a second or two of the observed reward. Thus, to make the repeated calculation of $e.\hat{h}$ computationally efficient, we maintain a window of recent time steps, removing from the credit assignment module's memory those older time steps that have less than ϵ_p potential probability of being targeted by future rewards. In other words, a time step e is removed once $P(targets(h, e)) < \epsilon_p$ for all possible h such that $h.t > t_{curr}$ where t_{curr} is the current time (i.e. for any future h).

One issue with delay-weighted, aggregate reward is that we do not know its value until the corresponding e passes through the window. Waiting for the final

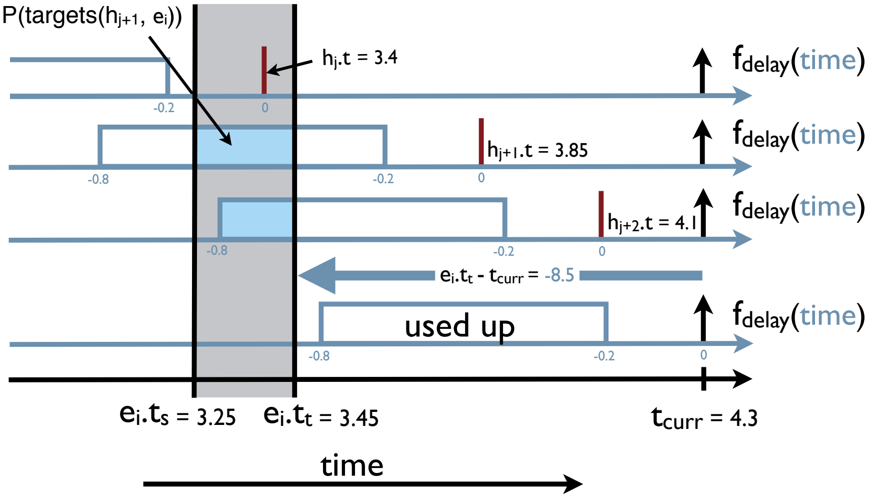


Figure 3.5: An illustration of an example calculation of $e_i.h$, using the start and stop times of e_i ($e_i.t_s$ and $e_i.t_t$), the times and values of three reward signals h_j, h_{j+1}, and h_{j+2}, and a Uniform(-0.8 seconds, -0.2 seconds) distribution for feedback delay density, f_{delay}. At top, the three probability calculations are shown for specific rewards, the timing of which is shown by the short red vertical bars (black in grayscale). At bottom, I show the calculation of how much probability mass is no longer available to future rewards (i.e., "used up"), based on the time difference between the current time t_{curr} and $e_i.t_t$. In this example, no probability mass is still available, so $\int_{t_{curr}-e_i.t_t}^{0} f_{delay}(x)dx = 1$ and the calculation of $e_i.h$ is complete; no future rewards can affect the value of $e_i.h$. In the plot, values and elements that are relative to a specific reward—rather than absolute time—or t_{curr} are in blue (or gray in grayscale). All values are in seconds. If we assume that all rewards have a value of 2, $e_i.h$ here is $(0 \times 2) + (0.333 \times 2) + (0.25 \times 2) = 1.167$ by Equation 3.1. If an intermediate $e_i.h$ had been calculated earlier when $t_{curr} = 3.9$ and probability mass was still available to future rewards, its extrapolated value would have been calculated as $\frac{\int_{-\infty}^{\infty}\int_{3.25-t}^{3.45-t} f_{delay}(x)\,dx\,dt}{\int_{-\infty}^{3.9}\int_{3.25-t}^{3.45-t} f_{delay}(x)\,dx\,dt}\left[(0 \times 2) + (0.333 \times 2)\right] = 1.143$ by Equation 3.3.

value creates a delay in learning that would be especially pronounced in long-tailed distributions. If the agent instead uses a human reward signal immediately—while still in areas of the state-action space that are similar to those targeted by the reward signal—to update $\hat{R}_H$, then it can quickly adjust its behavior, avoiding repetition of recent mistakes. We also expect such responsiveness to engage the trainer more effectively by immediately communicating the impact of feedback. However, learning from a state-action pair before its label is known does have trade-offs. The sooner the label is used for learning, even temporarily, the more it relies on possibly inaccurate predictions of the reward to come, adding noise or bias to the agents' behavior. Accordingly, our approach to learning from incomplete samples is parametrized to balance this trade-off.

To improve responsiveness to feedback, we generalize Equation 3.1 to create labels by extrapolating from what reward has been experienced already:

$$
\begin{aligned}
e.\hat{h} &= \frac{\int_{-\infty}^{\infty} \int_{e.t_s - t}^{e.t_t - t} f_{delay}(x)\,dx\,dt}{\int_{-\infty}^{t_{curr}} \int_{e.t_s - t}^{e.t_t - t} f_{delay}(x)\,dx\,dt} \sum^{h \in H_{hist}} h.v \quad P(targets(h, e)) && (3.2) \\
&= \frac{\int_{-\infty}^{\infty} \int_{e.t_s - t}^{e.t_t - t} f_{delay}(x)\,dx\,dt}{\int_{-\infty}^{t_{curr}} \int_{e.t_s - t}^{e.t_t - t} f_{delay}(x)\,dx\,dt} \sum^{h \in H_{hist}} h.v \int_{e.t_s - h.t}^{e.t_t - h.t} f_{delay}(x)\,dx && (3.3)
\end{aligned}
$$

The definite integral $\int_{-\infty}^{t_{curr}} \int_{e.t_s - t}^{e.t_t - t} f_{delay}(x)\,dx\,dt$ in the denominator expresses how much total targeting probability is "used up" for time step e. The total targeting probability, expressed in the numerator, is the integral over all possible reward times of the probability of targeting e. Thus, to extrapolate from a step that still has remaining target probability, we multiply the accumulated probability-weighted reward by the inverse of the proportion of total targeting probability that is no longer available to future rewards. Figure 3.5 illustrates the calculation of $e.\hat{h}$. This formulation inherently assumes that human reward within small windows of contiguous time steps generally correlates highly enough that recent reward yields a good approximation of immediately forthcoming reward. This generalized form of $e.\hat{h}$ allows reward to be immediately incorporated into $\hat{R}_H$, creating intermediary $e.\hat{h}$ values for temporary samples, which are replaced as newer $e.\hat{h}$ values are calculated. Thus flaws in this approximation have only a short-term effect on behavior.

This approach trades responsiveness for the added noise of extrapolating from unfinished steps. The trade-off is balanced by creating a threshold parameter c_{min} that determines the proportion of total targeting probability that must be "used up" before these intermediate samples are added to $\hat{R}_H$. In other words, a sample $e.t$ is used for learning only once $\int_{-\infty}^{t_{curr}} \int_{e.t_s - t}^{e.t_t - t} f_{delay}(x)\, dx\, dt > c_{min}$. A parameter of 0 maximizes responsiveness, and a parameter of 1 results in no extrapolation (i.e., Equation 3.1 can simply be used as a label) and thus no resultant noise. In informal testing, both extremes of c_{min} resulted in effective learning, but intermediate values gave the most subjectively satisfying balance between responsiveness and noise.

3.3.4 Implemented TAMER credit assignment methods

To date, two different credit assignment methods have been used in experiments of TAMER algorithms. The current method uses *delay-weighted, aggregate reward* (defined as $e.\hat{h}$ in Section 3.3.3) as the label for e.

Before finalizing this method, we conducted the mountain car experiments described in Section 3.4.2, in which the agent is assigned credit by what we term *delay-weighted, individual reward.* When comparing the current and prior techniques, I will refer to them more succinctly as "aggregate reward" and "individual reward", respectively. In the prior individual-reward method, the agent created a label for every possible pair of time steps (e's) and feedbacks (h's) for which $P(targets(h, e))$ is nonzero; thus each label corresponded to an individual reward, not a weighted sum of rewards as in the aggregate-reward method. For this individual-reward method, the label for a time step e and reward h was simply $h.v$ and the weight for that sample was $P(targets(h, e))$, giving a higher impact when there was a higher probability of h targeting e.[5] In contrast, the weights for samples

[5] In this discussion, I assume for clarity that sample weights are influenced only by the particular credit assignment method. However, other influences might be desirable, such as weighting more recent samples higher. We have not experimented with any of these other influences in our implementations.

under the aggregate-reward method are always 1.[6]

Both aggregate and individual reward techniques created qualitatively similar teaching experiences when tested informally by an author in mountain car. We changed to labeling by aggregate reward primarily for its qualitative characteristics. Specifically, when a trainer gives multiple rewards for a single time step e (a frequent occurrence), $e.\hat{h}$ aggregates multiple feedbacks into a single label. To illustrate why this trait is desirable, we consider an event e that results in 10 rapid +1 button presses under a corresponding interface. For the aggregate-reward method, $e.\hat{h}$ would roughly be 10 divided by the number of time steps that a single feedback signal might be targeting. Its weight would be 1. The older, individual-reward method would have simply yielded 10 samples with labels of 1 and weights ≤ 1; each label would be unrelated to the number of rewards given, a meaningful indicator of a trainer's intended intensity of feedback. Additionally, the individual-reward method would have created 10 times as many samples, increasing TAMER's computational load (though the extrapolation described in Section 3.3.3, if employed, may slow the aggregate-reward method).

Thus, labeling by aggregate reward is comparatively appealing for its ability to respond to multiple reward signals in an intuitively appropriate manner. However, our mountain car experiments were run before we saw the appeal of labeling by aggregate reward. Our analysis in Section 3.4.3 suggests that the empirical results would not differ dramatically had we used individual reward.

[6] The implemented version of individual reward differs further, in two less significant ways, from aggregate reward. Both differences reflect small refinements made since the mountain car data was collected. First, after each time step, a reward h used for credit assignment is created from the sum of manually delivered reward during that step, and $h.t$ is set to the end of the step. Second, in lieu of a method for stopping and starting training sessions, trainer engagement was determined by the presence of reward. Thus, any h such that $h.v = 0$ was not used to update $\hat{R}_H$. Therefore, unlike aggregate reward, individual reward does not allow learning from steps that have no probability of having produced reward signals.

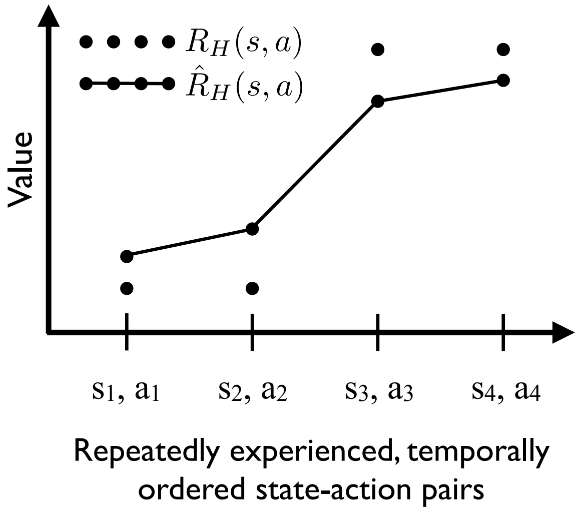


Figure 3.6: An illustrative example of temporal generalization from $e.\hat{h}$.

3.3.5 Characteristics of TAMER with delay-weighted, aggregate reward labels

As mentioned in Section 3.3.3, labeling samples for an unbiased estimator with aggregate reward does not guarantee that $\hat{R}_H = R_H$ with infinite samples. (Recall that R_H represents the expectation of human reward for any state-action pair, which I short-handedly call the human reward function, and $\hat{R}_H$ is the learned model of R_H.) I now explain a flaw of $e.\hat{h}$ labels and then discuss why we nonetheless use them in practice.

Since feedback for a time step generally contributes to the $e.\hat{h}$ calculation for temporally proximal time steps, an unbiased estimator over $e.\hat{h}$-labeled samples would exhibit temporal generalization. In other words, following the example in Figure 3.6, if some (s_2, a_2) is often experienced immediately before some (s_3, a_3), the learned values $\hat{R}_H(s_2, a_2)$ and $\hat{R}_H(s_3, a_3)$ will generally be nearer to each other than the true expectations $R_H(s_2, a_2)$ and $R_H(s_3, a_3)$. This temporal generalization occurs because the formulation of $e.\hat{h}$ makes its value generally dependent on those time steps before and after e.

Though it is easy to create pathological examples for learning $\hat{R}_H$ from $e.\hat{h}$-labeled samples, this aggregate reward method has been effective in practice. Further, Section 3.3.4 describes its desirable handling of the case of *multiple rewards targeting one step.* Alternatively, a single reward signal might be a response to a higher-level event made of multiple atomic time steps: *one reward targeting multiple steps.* Intuitively, $e.\hat{h}$ labels also accommodate violations of our assumption that each reward signal targets a point in time, since any one reward signal generally affects the labels for multiple recent time steps.

3.3.6 The full TAMER algorithm

Delay-weighted, aggregate reward ($e.\hat{h}$ labels) is incorporated into the infrequent-action TAMER algorithm (Algorithm 4) to create the full TAMER algorithm, shown in Algorithm 5. This extended algorithm uses additional notation introduced previously in this section.

If f_{delay} is a distribution with support given only to the previous time step (thus making f_{delay} dynamic if step duration varies), Algorithm 5 is equivalent to and thus a generalization of Algorithm 4. One exception is that I remove the concept of trainer engagement. Engagement is left out for descriptive clarity, since any distinct start/stop point would cut reward-gathering for at least one step, making the step incomplete. In our implementations, there is a button to end or start a training session, and the agent conservatively removes from memory any samples that are incomplete because of changes in engagement.

I now describe Algorithm 5 in detail. In accordance with our credit assignment technique, the start and end times and state-action pair of each step are recorded to the credit assignment window in lines 6 and 8. TAMER separates recording of the end time to allow the step to be on record before the action executes (when

Algorithm 5 The TAMER algorithm

Main thread

Input: ϵ_p

```
$i \leftarrow -1$
while $true$ do
  $i \leftarrow i + 1$
  $t_{curr} \leftarrow current\ time$
  if $i - 1 \geq 0$ then
    $e_{i-1}.t_t \leftarrow t_{curr}$
  end if
  $e_i.t_s \leftarrow t_{curr}$
  for all $e \in E_{hist}$ do
    if $P(targets(h, e)) < \epsilon_p$ for all $h \in H_{hist}$ such that $h.t > t_{curr}$ then
      $e.\hat{h} \leftarrow \sum^{h \in H_{hist}} h.v \int_{e.t_s - h.t}^{e.t_t - h.t} f_{delay}(x)dx$
      $updateModel(\hat{R}_H,\ (e.s, e.a, e.\hat{h}))$          // add completed sample
      $E_{hist} \leftarrow E_{hist} \setminus e$          // remove sample from memory
    end if
  end for
  for all $h \in H_{hist}$ do
    if $\int_{e.t_s - h.t}^{0} f_{delay}(x)dx = 1$ for all $e \in E_{hist}$ then
      $H_{hist} \leftarrow H_{hist} \setminus h$          // remove fully credited reward from memory
    end if
  end for
  $e_i.s \leftarrow getState()$
  $e_i.a \leftarrow argmax_a \hat{R}_H(e_i.s,\ a)$
  $E_{hist} \leftarrow E_{hist} \cup \{e_i\}$          // record new experience
  $takeAction(e_i.a)$
  wait for next time step
end while
```

Human interface thread

```
while $true$ do
  if $new\ reward\ h\ with\ value\ h.v\ is\ received$ then
    $h.t \leftarrow current\ time$
    $H_{hist} \leftarrow H_{hist} \cup \{h\}$          // add new reward signal
  end if
end while
```

the end time is still unknown), making it possible to credit a time step for reward that occurs before it finishes, which avoids the need to store rewards in memory.

In lines 9 to 15, the model is updated as follows. For each time step that cannot receive at least ϵ_p credit for a future reward (expressed in lines 9 and 10), the label $e.\hat{h}$ for the step's sample is calculated as in Equation 3.1 (line 11), the model $\hat{R}_H$ is updated with e (line 12), and e is removed from the agent's experiential memory (line 13). To keep explication simple, this description of the full algorithm does not extrapolate from time steps that have received credit but have not yet reached the threshold ϵ_p for removal (thus parameter c_{min} is absent). However, to add extrapolation, the agent updates $\hat{R}_H$ with unfinished samples containing extrapolated labels that are replaced as more complete labels become available. The extrapolation is done by Equation 3.3, as explained in Section 3.3.3. Specifically, to "replace" a sample e_i with its newer version, the model must be rolled back to before e_i was added without affecting the impact of other samples that should not be removed. A general solution is to maintain a "forward-only" variant of $\hat{R}_H$ that is learned from only finished samples that never need be removed and also keep a temporary $\hat{R}_H$ that is constructed each time step from the forward-only version—adding all unfinished samples each round—and is used for action selection.

Once the update of $\hat{R}_H$ occurs, lines 16–20 of the algorithm remove from H_{hist} any reward signals that can no longer impact the labels for future time steps or steps still active in the credit assignment module (in E_{hist}). These removals save memory and reduce the computational cost of line 11. The algorithm then proceeds equivalently to Algorithm 4, except also adding each new experience e_i to the set of experiences in memory, E_{hist}, at line 23.

In the human-interface thread, looping constantly, newly received reward is added to H_{hist}.

3.4 TAMER instantiations

The full TAMER algorithm was defined in Section 3.3.6. However, in order to instantiate the algorithm in a particular domain, six components need to be specified:

- the representation of $\hat{R}_H$, which implicitly includes any feature extraction from state and action;
- the updateModel() method that learns $\hat{R}_H$ given a stream of incoming step-reward samples;
- f_{delay}, the probability density function for delay from an event to its resultant reward;
- ϵ_p, the threshold for considering a sample "finished";
- c_{min}, the extrapolation parameter that balances responsiveness and action noise; and
- the training interface, including a binary definition of trainer engagement.

We have successfully implemented TAMER on 5 task domains.[7] The corresponding algorithms have been evaluated with different focuses. In this chapter, I first present results on two contrasting task domains—Tetris and mountain car—using multiple human subjects as trainers. For these two tasks, I focus on the comparative performance of TAMER agents and agents learning without teaching (from hard-coded reward). Another task domain is interactive robot navigation. For this domain, I focus on the flexibility of TAMER to learn multiple, qualitatively different policies when the algorithm differs only by what reward it receives from the trainer. I give proof-of-concept results for which the author is the trainer,

[7] In addition to these 5 task domains, Sridharan employed TAMER to teach policies in 3v2 keep-away soccer[87].

demonstrating TAMER's applicability to robots. For each implementation, I provide a bulleted description corresponding to the above six components.

Chapter 4 describes a TAMER implementation for a balancing cart pole task, and Chapter 6 includes a TAMER algorithm for a grid world task as well as an algorithm that generalizes TAMER.

For readability, in the following sections I separate algorithmic descriptions and results for Tetris and mountain car from those of the interactive robot navigation task; the experiments for the first two domains focus on comparing TAMER and autonomous algorithms, whereas the results from the robot task demonstrate the flexibility of a single TAMER implementation to learn multiple behaviors.

3.4.1 Instantiations for comparison to RL algorithms

In the Tetris and mountain car domains, a full MDP is specified to compare the TAMER agent learning within the corresponding MDP\R to reinforcement learning agents learning within the MDP. Though the TAMER agent actively seeks to maximize predictions of human reward (by $\hat{R}_H$), the true objective of the human-agent system is to maximize some task performance criterion, whether it be the trainer's subjective evaluation or an explicitly defined metric. For this reason, we define the task performance metric τ to be the same metric as that of the RL agents, the cumulative MDP reward that would be received throughout all episodes in a run. In each domain, R (unobserved by the TAMER agent) was easily communicated to the trainer, who was effectively instructed to train his or her agent to earn the highest sum of environmental reward per trial episode. Also, in each domain the state and action are displayed graphically to the human trainer. Both task domains were adapted from implementations within RL-Library [94].

The first domain, Tetris,[8] is a puzzle computer game in which pieces of

[8] Our Tetris specification follows that described by Bertsekas and Tsitsiklis [5].

various shapes (called “tetrominos”) fall from the top of the screen, and the player’s task is roughly to fit each piece with previous pieces below to make solid horizontal lines. Each such line disappears upon filling its last hole(s), and the pieces above move down one position. Play ends when a piece cannot be placed because previous pieces are stacked too high, and the object of our specification of Tetris is to clear as many horizontal lines as possible before play ends. Tetris has a large state space, a variable describing the type of falling tetromino piece and a vector of 200 boolean variables describing which cells of the game board are filled. Following the precedent of all other work on Tetris cited in this chapter, we consider the final placement of a falling Tetris piece to be a single action. Thus the number of possible actions per time step is variable, and the time step frequency in Tetris—less than one action per second—allows human trainers to easily reward individual actions. For each piece placement, the MDP reward is the number of lines cleared during that action, from 0 to 4. For the second Tetris experiment only, we made two important changes to the RL-Library version of Tetris. First, after positive reward the screen immediately flashes a transparent green color. After negative valued reward (i.e., punishment), the screen flashes red. These flashes were added to improve the trainer’s confidence that her reward is being received. Secondly, the most recent piece to be placed is drawn in black to make clear that feedback can currently be given on that specific piece’s placement.

In the second domain, mountain car, a simulated car must get to the top of a hill. The car begins between two steep hills and must go back and forth to gain enough momentum to reach the goal. Mountain car has a continuous, two-dimensional state space; its variables are position and velocity. In the specification we use, three actions are available: $+c$ acceleration, $-c$ acceleration, and no acceleration. At each time step except the final step at the goal, the MDP reward is -1. In our experiments in this chapter, actions occurred approximately every 150 milliseconds, preventing trainers from labeling specific actions. Also, we modified the RL-Library task to show the agent's action to the trainer and to start agents each episode in a random location within bounds around the bottom of the hill.

These two domains complement each other. Tetris has a complex state-action space and low time step frequency, whereas mountain car is simpler but occurs at a high frequency.

Tetris Instantiation

We conduct two experiments of training TAMER agents to play Tetris in Section 3.4.2, each with similar instantiations. These instantiations of TAMER on Tetris are implementations of infrequent-action TAMER (Algorithm 4). Thus, for the full time that the agent places the current piece, the trainer can give reward to the *previous* piece's placement. The components of TAMER are specified as below.

- The *representation of* $\hat{R}_H$ is a linear model over the state-action features. These features are described below, including the expanded feature set used in the second experiment.

- The *updateModel() method* is incremental gradient descent. The step sizes for during the first and second experiments were respectively 0.0000025 and 0.000005 divided by the number of features.

- f_{delay}, ϵ_p, and c_{min} are not applicable for Algorithm 4.

- For the *training interface*, 'p' and 'n' keys respectively gave +1 and −1 reward in the first Tetris experiment, and '/' and 'z' keys do the so in the second Tetris experiment. In the second experiment, total reward for a placement—creating a single label—was limited to the range $[-4, 4]$. A computer screen displays the Tetris game in action. A trainer is considered engaged during a time step if and only if the trainer gives a non-zero label for that step.

To create the state-action features that are input to $\hat{R}_H$, state features are first extracted from both the current state and the deterministic next state (before the new Tetris piece appears). The next state is merely the previous state with the new blocks in place—which the action dictates—and any full horizontal lines removed. The difference between these two state-feature vectors create the state-action features that are input to $\hat{R}_H$. The state features are (with those unique to the second experiment italicized)

- the 10 column heights;
- the 9 differences in consecutive column heights;
- the maximum column height;
- the number of holes, defined as empty spaces below blocks;
- *the sum of well depths, where a well is a vertical column of empty space, the top space of which has blocks or Tetris-board borders immediately to the left and right and only empty space above;*
- *the maximum well depth; and*
- *the 23 squares of the previous 23 features.*

Because the squares of column heights (and the maximum column height) can reach relatively large values, we divide these features by 40, a value chosen to ensure the corresponding state-action features stay roughly within in the range $[-3, 3]$. Also, a constant feature of value 1 is added to the state-action features. The features that were used in both experiments were developed by Bertsekas and Tsitsiklis [5], and the sum of well depths and the maximum well depth have also been used previously to specify Tetris policies [27, 9]. The squared features are novel aspects of our representation. Feature extraction is illustrated in Figure 3.7.

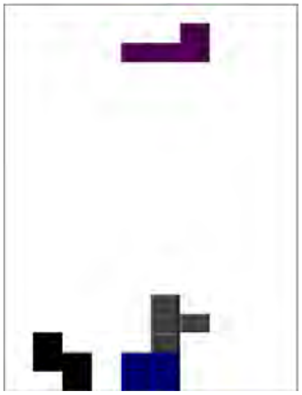


Figure 3.7: A screenshot of Tetris under random control, where the previous piece placement is blackened. The extended state-feature extraction would result in the column heights $(0, 3, 2, 0, 2, 5, 4, 0, 0, 0)$, differences in consecutive column heights $(3, -1, -2, 2, 3, -1, -4, 0, 0,)$, maximum column height 5, 4 holes, sum of well depths 5, the maximum well depth 3, and the 23 squares $(0, 9, 4, 0, 4, 25, 16, 0, 0, 0; 9, 1, 4, 4, 9, 1, 16, 0, 0; 25; 16; 25; 9)$ of the previous features. State-action features are calculated from state features; the state-action features for the previous placement are the previous state features subtracted from these state features.

Mountain car instantiation

For mountain car, we implemented the full TAMER algorithm as described in Algorithm 5, except that it employs our earlier credit assignment technique, delay-weighted individual reward, which was explained in Section 3.3.4.[9]

- The *representation of* $\hat{R}_H$ is a linear model over features defined as Gaussian radial basis functions, described below.
- The *updateModel() method* is incremental gradient descent with a step size of 0.001.
- f_{delay} is a Uniform(-0.8 seconds, -0.2 seconds) distribution.
- ϵ_p and c_{min} are not applicable for delay-weighted individual reward.

[9] For a line-by-line description of this mountain car algorithm with delay-weighted individual reward, see [47].

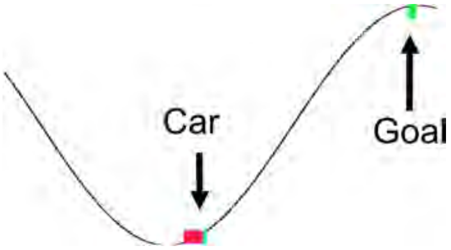


Figure 3.8: A screenshot of the mountain car task. The action is shown by location of the light blue vertical bar on the left (accelerating left), center (no acceleration), and right (accelerating right, as in this screenshot) of the car.

- the training interface: 'p' and 'n' keys respectively give +1 and −1 reward. A computer screen displays car and its most recent direction of acceleration as in Figure 3.8. A trainer is considered engaged during a time step if and only if the trainer gives a non-zero label for that step.

The features for $\hat{R}_H$ are 2-dimensional Gaussian radial basis functions over state, one set per action, following the specification given by Sutton and Barto [90]. Specifically, for each action, the state space is divided equally into a 15 × 15 grid of RBFs, with means ranging from the minimum to the maximum possible values of each state dimension. The squared "width" of each RBF, σ^2, is 0.08, where the unit is the distance in normalized state space between adjacent Gaussian means[10] For each action, there is also a feature with value 0.1 when that action is input and 0 when it is not.

3.4.2 Comparative results and discussion

Our experiments in this section test three hypothesized benefits of interactive shaping via TAMER: (1) compared to agents learning from MDP reward, shaping decreases how many samples are needed to learn a "good" policy, (2) an agent can learn in the absence of a coded evaluation function, and (3) lay users can designate

[10] Normalization linearly maps the value of each state variable to [0,1].

behavior effectively (unlike programmed behavior). For the experiments, human trainers observed the agents in simulation on a computer screen.

Human reward was given via two keys on the keyboard, which mapped to 1 and -1. This mapping was chosen for two reasons. First, making the outputs of the keys either both positive or both negative would create a counterintuitive training interface. For example, mapping both keys to a positive value would make multiple "punishment" key presses more desirable than a single "punishment" key press, a clearly undesirable outcome. Second, given that a "reward" key maps to a positive value and a "punishment" key maps to a negative value, only the relative scaling of the two values matters; i.e., one value is $-c$ times the other. Though this scaling constant c is a worthwhile direction for future investigation, the most effective value of c is not a dimension of this chapter's investigation, and we use the most obvious and intuitive value, $c = 1$. This mapping is the same as or similar to that of related works that explain their exact mappings [105, 101, 76].

The trainers were read instructions—including that keys could be pushed multiple times for stronger effect—and were not told anything about the agent's features or learning algorithm. In our initial Tetris experiment, we gathered data from 9 human trainers. In a later experiment, 27 subjects trained Tetris agents that used a different state-action feature set. 19 human subjects trained mountain car agents. In both the first Tetris experiment and the mountain car experiment, at least a fourth of the trainers did not know how to program a computer. Experimental instructions and videos of agents before, during, and after training can be found online.[11]

[11] `http://www.cs.utexas.edu/~bradknox/papers/tamer`

Tetris

Tetris is notoriously difficult for temporal difference learning methods that model a value or an action-value function (in contrast to those that maintain only a policy representation), especially when the algorithm updates incrementally (after each action, using only the most recent learning sample). In our own work, we were able to get SARSA(λ) [90] to clear approximately 30 lines per game at best, using a very small step size α and running for hundreds of games. Bertsekas and Tsitsiklis [5] report that they were unable to get optimistic TD(λ) to make "substantial progress". RRL-KBR [81] does somewhat better, getting to 50 lines per game after 120 games or so. I show this result, the results of other Tetris learning algorithms, and results from our first Tetris experiment in Table 3.1.[12]

Algorithms that update a value function from a batch of samples fare better. Two of the three such algorithms of which we are aware reach a peak performance of clearing several thousand lines per game [5, 28]. However, both algorithms' performances drop substantially from their peak, descending to less than half of their maximum. Indeed, this "unlearning" occurs with all but one of the value-function-based techniques for which such results are reported. The one exception is the third algorithm to use batch updates to learn a value function, an approximate value iteration algorithm that uses a discount factor [75] below what is correct for the Tetris MDP. This algorithm achieves a non-zero score (according to a visual assessment of the corresponding plot) after approximately 2000 games.

Policy search algorithms, which do not model a value or reward function, attain the best final performance, reaching at least hundreds of thousands of lines per game [9, 92, 11, 103], though they require many games to reach such performance and do not learn within the first 100 games. Notably, the linear function and

[12] We should note that Ramon et. al. rejected a form of their algorithm that reached about 42 lines cleared on the third game. They deemed it unsatisfactory because it unlearned by the fifth game and never improved again, eventually performing worse than randomly. Ramon et al.'s agent is the only one we found that approaches the performance of our system after 3 games.

corresponding features used by Szita and Lorincz are identical to that of Bertsekas and Tsitsiklis' batch algorithm, λ-policy iteration. Experimental investigation undertaken by Kalyanakrishnan [41] suggests that whereas the linear representation introduced by Bertsekas and Tsitsiklis can support high-valued policies, the same representation does not provide good enough approximations of value functions to stably learn high-valued policies.

In our first experiment, 9 subjects—graduate students and acquaintances of the authors—first practiced for two runs. Data from the third run is reported and compared to the performance of other Tetris-learning algorithms. Trainers were given 10 games to train their agents and were allowed to quit once they thought their agent had reached peak performance, after which the agent continued to run with the learned, static policy. Of the algorithms that incrementally model an actual function, TAMER learns most quickly and to the highest final performance, reaching a mean of 65.89 lines per game during the third game (Table 3.1 and Figure 3.9).[13] In this first experiment, TAMER agents learn much more quickly than all previously reported agents and reach a final performance that is higher than all other incrementally updating algorithms.

In our second experiment, 27 trainers from the general undergraduate population at the University of Texas at Austin were given 45 minutes to train an agent after one 3 minute practice.[14] This time limit differs from the episode limit used in the first experiment. With this data, we examine performance of the learned policy after intervals of 80 time steps (i.e., piece placements). Each static policy is tested over 20 episodes and the mean is considered one subject's sample at that interval.

[13] Most trainers stopped giving feedback by the end of the fifth game, stating that they did not think they could train the agent to play any better. Therefore most agents are operating with a static policy by the sixth game. Most score variations come from the stochasticity inherit in Tetris, including the highest scoring game of all trainers (809 lines cleared), which noticeably brings the average score of game 9 above that of the other games.

[14] The data for this experiment is also presented as the Teaching condition of the critique experiment in Chapter 5. The experiment was a collaborative effort by Brian D. Glass, Bradley C. Love, W. Todd Maddox, the author, and his advisor.

Table 3.1: Results of various Tetris agents. The four algorithms immediately below TAMER use a value function for action selection. The policy function for the other five algorithms is not created by modeling values.

Method	Mean lines cleared		Games for Peak
	in game 3	at peak	
TAMER	**65.89**	**65.89**	**3**
λ-policy iteration [5]	no learning until game 100	3183	1500
RRL-KBR [81]	~5	~50	120
Approx. dynamic programming [28]	no learning until game ~360,000	3,500-5,000	unreported
Approximate value iteration [75]	no learning until game 100	20,000-25,000	~6000
Natural policy gradient [40]	no learning until game 100	5,000-7,000	~6000
Genetic algorithm [9]	~0 (no learning until game 500)	586,103	3000
Noisy cross entropy [92]	~0 (no learning until game 100)	348,895	5000
CMA-ES [11]	~0 (no learning until game 1500)	36,000,000	~22,500
Noisy cross entropy [103]	~0 (no learning until game 100)	36,000,000	500

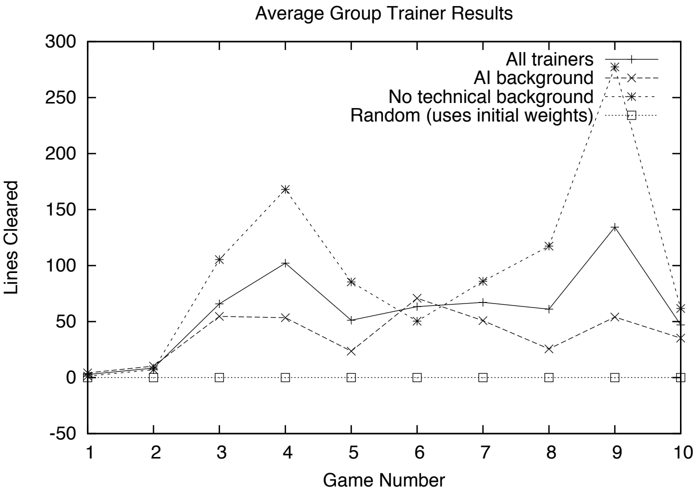


Figure 3.9: The mean number of lines cleared per game by each experimental group in our first Tetris experiment.

Using time steps instead of episodes gives another perspective on learning speed, which is especially important for Tetris, where episodes can last tens or many thousands of time steps. For this experiment, trainers were stopped without finishing their last episode. The last episode was often much longer than the others, since it had been prolonged by the agent beginning to play well. Thus the first episode to feature high performance in this data often remains unfinished, and so we do not examine performance during training. Additionally, unlike in the first experiment, trainers were not told to stop training when satisfied with their agent's performance. The TAMER agent in this second experiment uses a larger and improved set of features, specified in Section 3.4, that achieves higher peak performance. In Figure 3.10, I present agent performance by interval and the observed distribution of performance after the last interval test that all subjects completed. The mean performance after this final common interval is 618.32 lines per game with a standard error of 130.91. If we instead look at the final interval for each subject, which may go beyond 720 time steps of training, the mean rises to 770.8741. This improvement of approximately 10 times the number of lines cleared in the first experiment shows that TAMER's performance is unsurprisingly connected to the representation of $\hat{R}_H$. Because games played by well-trained policies last so long, the trainers who found success early did not finish many games. Indeed, one trainer did not finish a single game; six did not finish three games. Therefore, I do not report performance by game.

Additionally, of the Tetris algorithms that model an actual function, our gradient descent, linear TAMER algorithm is one of two algorithms that clearly do not unlearn in Tetris. The other algorithm, by Petrik and Scherrer [75], also learns with a discount factor below that of the MDP. We suspect that both algorithms' low reliance on bootstrapping—via low discount factors—reduces their sensitivity to their model's approximation error. Discounting future human reward is discussed in Section 2.5.1 and extensively examined in Chapter 6.

During this second experiment, subjects were prompted, "If you have played Tetris before, please rate your Tetris playing skills." They answered on 1–7 Likert scale, where higher numbers correspond to better skills. The Pearson correlation between self-rated playing skill and final agent performance (after 720 tetromino placements) is weakly positive, $r(35) = 0.09$, and insignificant, $p = 0.69$. This result suggests that the ability to perform a task may be only weakly related to the ability to teach the task through interactive shaping. If so, then shaping would be an especially favorable teaching method when the teacher lacks the expertise to provide quality demonstrations. However, the conjecture that expertise is not needed for interactive shaping needs further examination.

Mountain car

The Tetris results demonstrate TAMER's effectiveness in a domain with actions that are infrequent enough for a trainer to precisely target them with reward. In contrast, our experiments in the mountain car task test TAMER's performance in a

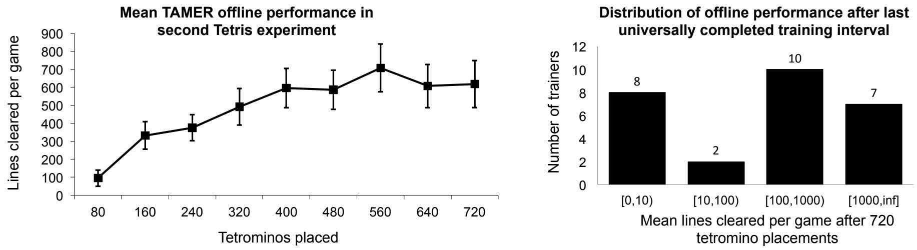
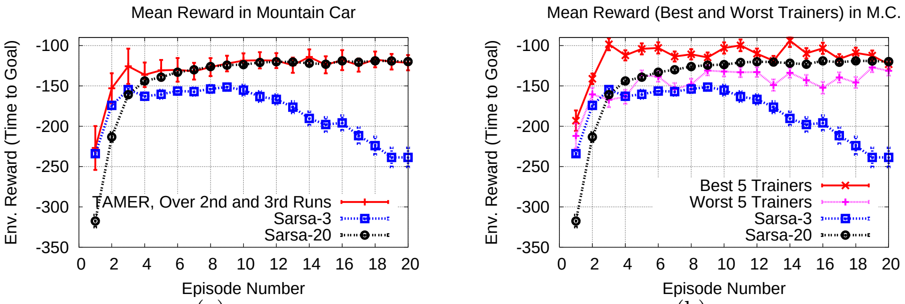


Figure 3.10: Results from the second Tetris experiment. The figure on the left shows performance in offline testing of the learned model $\hat{R}_H$ at increments of 80 tetromino placements. Error bars express standard error. On the right, the distribution of performance after 720 time steps of training.

domain with quick, frequent actions. We compare TAMER to the reinforcement learning algorithm SARSA(λ), a common RL algorithm that has been applied successfully to mountain car in the past [90]. Here, SARSA(λ) learns from MDP reward with the same function approximator as the TAMER algorithm (a linear model over Gaussian RBF features, using gradient descent updates). Two SARSA(λ) agents were used: one tuned for total cumulative MDP reward across all previous episodes (Figure 3.12) after 3 episodes and one tuned for after 20 episodes, which I will respectively refer to as Sarsa-3 and Sarsa-20. We tuned via a hill-climbing algorithm that varied one parameter (α, λ, or ϵ in the standard notation of SARSA(λ) [90]) at a time, testing the agent's performance under each value for that parameter, taking the best-performing value, and then repeating (for fifty or more iterations).[15] The specific number of episodes, 3 and 20, were chosen to represent different emphases on the trade-off between learning quickly and reaching the best asymptotic

[15] The parameters resulting from tuning for Sarsa-3 are $\alpha = 0.835$, $\lambda = 0.592$, and $\epsilon = 0.00000879$. For Sarsa-20, the parameters are $\alpha = 0.424$, $\lambda = 0.384$, and $\epsilon = 0$.

For both algorithms, the action-value function Q is represented by a linear model over Gaussian RBF features. For each action, 1600 RBF means are located on a 40×40 evenly space grid over the state space, where the outermost means in each dimension lie on the extremes of the dimension. Additionally, an activation feature of 0.1 is added for each action, creating a total of 4803 state-action features. When an action is input to Q, the features for all other actions are zero. The squared width σ^2 of the Gaussian RBFs is 0.2, following Sutton and Barto's definition of an RBF's "width" and where the unit is the distance in normalized state space between adjacent Gaussian means. The weights of Q are all optimistically initialized to 0. ϵ was annealed by 0.99 at the end of each episode.



(a) (b)

Figure 3.11: Error bars show a 95% confidence interval (with an assumption that performance between human-agent teams is distributed normally and each run is an independent sample). (a) The mean MDP reward (-1 per time step) received for the mountain car task for the second and third agents (i.e., training runs) shaped by each trainer under TAMER and for SARSA(λ) agents, using parameters tuned for best cumulative reward after 3 and 20 episodes. (b) Mean performance of agents shaped by the best five and worst five trainers, as determined over all three runs.

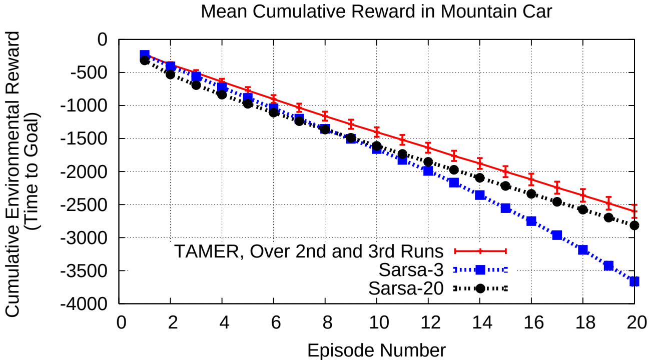


Figure 3.12: Mean cumulative MDP reward received for the mountain car task.

performance.

For this mountain car experiment, 19 human subjects—also graduate students and acquaintances of the authors—trained agents. The TAMER agents were shaped for three runs of twenty episodes by each trainer. We consider the first run a practice run for the trainer and present the combined data from second and third runs. Results are shown in Figures 3.11 and 3.12. Figure 3.11(a) shows that the

TAMER agents, on average, consistently outperformed the Sarsa-3 agents and outperformed the Sarsa-20 agents for the first five episodes, after which Sarsa-20 agents showed comparable performance. Under the guidance of the best trainers (Figure 3.11(b)), TAMER agents consistently outperform any Sarsa agent, and, under the worst trainers, they perform somewhat worse than the Sarsa-20 agent. Importantly, TAMER agents, on average, also outperformed each Sarsa agent in mean cumulative MDP reward through the length of a run (Figure 3.12). Since each time step incurred a -1 MDP reward, Figure 3.12 is also a measure of sample size. The four trainers who did not have a computer science background achieved mean performance as good or marginally better than the fifteen who did. Overall, the results, though less dramatic than those for Tetris, support our claim that TAMER can produce faster learning than algorithms that learn from MDP reward.

3.4.3 Additional results

Having presented the main experimental results in Sections 3.4.2 and 3.4.2—focusing on TAMER's comparative effectiveness and its accessibility to non-programmers—I now present further analysis of the data gathered from the Tetris and mountain car experiments. Section 3.4.3 presents an analysis of training patterns, specifically looking at the frequency of feedback and the ratio of positive to negative reward. Then Section 3.4.3 examines the consequences of assigning credit by individual reward versus aggregate reward (first discussed in Section 3.3.4).

Patterns among trainers

For both Tetris and mountain car, I now describe patterns involving the frequency of feedback and the ratio of positive to negative reward. For each, we look at how they change over time and their relationship to performance. The Tetris data comes from the larger, second experiment with 27 subjects. In this section, performance in mountain car is measured online, during the duration of training, whereas performance in Tetris is tested offline after 720 tetrominos have been placed, as I describe in Sections 3.4.2 and 3.4.2 respectively.

We calculate the ratio of positive to negative reward over a set of reward instances H_{hist} as

$$\frac{1 + \sum\limits^{h \in H_{hist} | h.v > 0} h.v}{1 - \sum\limits^{h \in H_{hist} | h.v < 0} h.v}.$$

The sums of positive and negative reward are each padded by one to prevent the numerator and the denominator from being zero. As has been observed previously [109], trainers gave more positive reward than negative reward over the entire training run (720 piece placements in Tetris and 20 episodes in mountain car). This finding is quite consistent; the one exception among the 46 trainers examined here is a mountain car trainer whose ratio is exactly 1. Note that the 8 Tetris trainers whose agents could not clear even 10 lines a game (Figure 3.10)—in many cases averaging less than a line cleared per game—still all gave more positive than negative reward. We also find that the ratio of positive to negative reward over the entire run has a significant and strong positive correlation with performance: $[r(25) = 0.45, p = 0.02]$ for Tetris and $[r(17) = 0.57, p = 0.01]$ for mountain car. Additionally, the ratio generally increases with time, with the balance tipping further towards positive reward as training progresses; a paired t-test comparing the ratio during the first 160 tetromino placements to the ratio during tetromino placements 561–720 reveals a significant increase $[t(26) = -10.12, p < 0.0001]$, and a comparison of the first 5 episodes of mountain car training to episodes 16–20 also shows a significant increase $[t(18) = -2.77, p = 0.01]$.

Feedback frequency was calculated as the proportion of time steps with non-zero reward. In our experiments, feedback frequency drops with time; paired t-tests over the same training intervals show significant decrease: $[t(26) = 2.46, p = 0.02]$ for Tetris and $[t(18) = 3.05, p = 0.007]$ for mountain car. These observations agree with those of Isbell et al. [36] but contrast with those by Thomaz and Breazeal [109]. In mountain car, feedback frequency over the entire training run correlates significantly with performance $[r(17) = 0.52, p = 0.02]$, a strong positive relationship. This correlation also holds for the frequency over only the first episode and performance (still over the full training run) $[r(17) = 0.49, p = 0.03]$. In Tetris, however, feedback frequency does not significantly correlate with performance: $[r(25) = -0.06, p = 0.75]$ for full-run frequency and $[r(25) = -0.22, p = 0.27]$ for first-episode frequency.

The causal relationships between these variables is unclear. For instance, the positive correlation between performance and positive-to-negative reward ratio could be explained by multiple hypotheses: that higher performance earns higher reward, that the trainers who give more positive feedback give better feedback, some third hypothesis, or by the combined effects of these hypotheses. Therefore, these results suggest future directions of controlled experimentation to test various hypotheses that might guide the choice among and preparation of trainers. Further, we find a highly consistent pattern of trainers giving more positive than negative reward. This observation is fundamental to the analysis in Chapter 6.

Impact of credit assignment

Though we do not expect that our change in credit assignment techniques from delay-weighted, individual reward to delay-weighted, aggregate reward will create a large performance difference, we nonetheless quantitatively test this difference on data that is already on hand. To this end, we used the 19 training logs from the third runs of our mountain car experiment and tested the static policy learned from each log, allowing a maximum of 500 steps per episode before restarting the episode, under a number of different conditions: the number of episodes the agent trained on

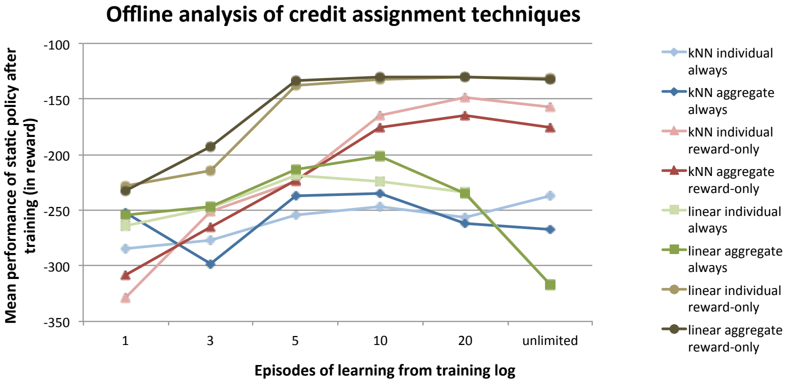


Figure 3.13: A comparison of aggregate and individual reward credit assignment techniques. For legend entry "x y z", x is the $\hat{R}_H$ model type, y indicates whether aggregate or individual crediting was used, and z indicates whether all samples were used to learn $\hat{R}_H$ or only samples with non-zero labels were used.

the log (the x-axis); whether $\hat{R}_H$ was learned and represented by k-nearest neighbors, where k is the square root of the current sample size, or by a linear model using 2-dimensional radial basis features (RBFs) and incremental gradient-descent updates; and whether updates occurred always or only when reward had been received. We additionally test between the conditions of whether credit assignment uses individual or aggregate reward. This final condition is of chief interest. We ask, when the other conditions are held constant, does crediting with individual or aggregate reward produce better performance?

The condition of whether updates occur only when reward is received deviates from our current strategy—always update when training—because the data was gathered without trainers having the ability to toggle whether they are currently training; trainer engagement instead was determined by the presence of reward during a time step, yielding "reward-only" updates. For individual rewards, reward-only updates only occur on time steps during which non-zero reward was received, following the algorithm used online during data gathering as described by Knox and Stone [47]. For aggregate rewards, reward-only updates can be implemented variously, since a time step's sample is made over multiple subsequent time steps, some of which may receive reward while others do not; in these experiments, reward-only updates only occur with aggregate rewards when the sample has non-zero aggregate reward.

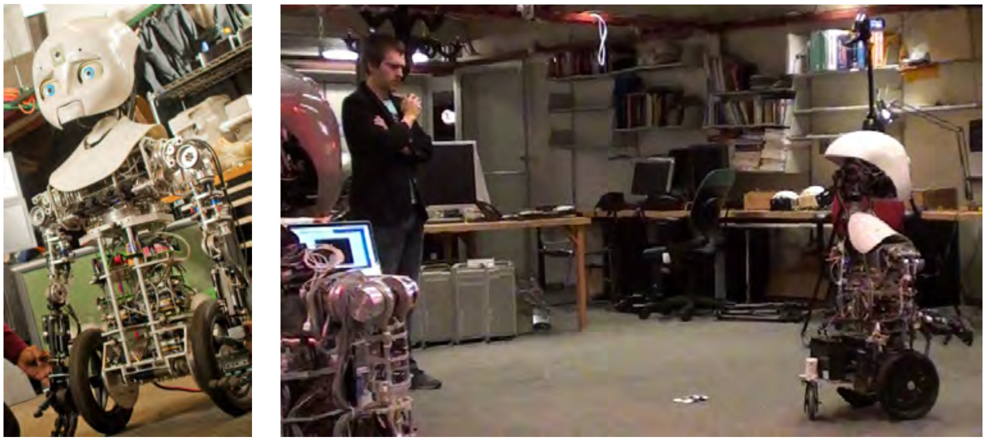
In this analysis, there is no clear winner. In the highest-performing pairing, the aggregate method is slightly better overall. But in another pairing, individual credit is typically a bit better. And in the other two pairings, the superior method changes with the number of episodes of learning. Most importantly, the results across the pairings are all close.

Considering that the training occurred in the first place under individual, reward-only learning with a linear model and that trainers adapt to the specific learning algorithm, these results are positive for the aggregate method. In other words, this offline analysis suggests that the aggregate method is at least as good as the individual method. Considering further that the aggregate method is qualitatively more elegant and results in far fewer samples to be processed or stored—indeed, of the 10 or so slowest experimental instances to complete (of the 912 tested), all were of the individual method with unlimited episodes of learning—the aggregate method appears superior. Thus, TAMER now uses aggregate credit assignment.

To fully test which method is better would require online training with both methods in several learning domains. However, because the change in the credit assignment method is relatively minor, we are satisfied with this offline analysis.

3.4.4 Instantiation and results for interactive robot navigation

Here I present an instantiation of TAMER for a third task, interactive robot navigation. Unlike the experiments reported in Section 3.4.2, the results in this section aim to demonstrate the flexibility of TAMER to learn multiple interesting behaviors, each



(a) (b)

Figure 3.14: (a) The MDS robot Nexi. (b) A picture of Nexi being trained by the author. The artifact used for trainer interaction can be seen on the floor immediately behind Nexi, and the trainer holds a presentation remote by which reward is delivered.

in a different training session, without changing the algorithmic instantiation.[16]

In this task, a Mobile-Dextrous-Social (MDS) robot named Nexi can sense the location and orientation of itself and a training artifact. The artifact is a small object that can be easily moved by the trainer. The robot's actions are to move forward, turn left, turn right, or stay still. Using its sensory data as context, the robot is taught a number of qualitatively different behaviors. Pictures of Nexi and a training session are shown in Figure 3.14.

This task differs from all other tasks for which TAMER is currently implemented in three important ways.

1. *The agent is a physically embodied mobile robot.* Much of TAMER's motivation involves robotic agents. However, learning and acting in the physical world—a challenging environment to sense—brings new difficulties; to say that TAMER is applicable to robots, we should show that it works on an actual robot.

[16] The research described in this section was conducted at the MIT Media Lab in collaboration with both Cynthia Breazeal and Peter Stone.

2. *The task is interactive.* In other TAMER tasks, the training is interactive, but the task itself is not; in this task, the environmental state can be directly changed by the trainer. We expect that many personal and service robots will operate in environments in which humans appear as relevant state, such as with human-robot collaboration on an assembly line. Only this task captures such a scenario.

3. *The task environment is designed with the expectation that multiple interesting behaviors can be taught and performed in it.* Other tasks and TAMER implementations certainly can accommodate multiple qualitatively different behaviors, but they were designed with a single task performance metric in mind.

Though evaluation in this domain is limited with respect to who trains the TAMER agent (the author), multiple target behaviors are trained, giving the evaluation a different dimension of breadth than the previous experiments. Indeed, a primary motivation of implementing TAMER for this task is to demonstrate its ability to flexibly learn different behaviors.

The robot estimates its environmental state through a Vicon Motion Capture system that determines the 3-dimensional locations and orientations of the robot and the training artifact; in estimating its own position and orientation, the robot employs dead reckoning, using both the Vicon data and information from its wheel encoders. The duration of time steps varies by the action chosen. Moving forward and staying last 1.5 seconds, whereas turns occur for 2.5 seconds. When moving forward, Nexi attempts to move at 0.075 meters per second, and Nexi seeks to turn at 0.15 radians per second. Since changes in intended velocity—translational or rotational—require a period of acceleration, the degree of movement during a time step was affected by whether the same action had occurred in the previous time step.

The MDS robot Nexi

The Mobile-Dextrous-Social robot platform is designed for research at the intersection of mobility, manipulation, and human-robot interaction [16]. It was built by a team led by Cynthia Breazeal at MIT that includes the Laboratory for Perceptual Robotics at the University of Massachusetts at Amherst, Xitome Design, Meka Robotics, and digitROBOTICS.

The mobile base of the MDS platform has 2 degrees of freedom, with two powered wheels and one unpowered, stability-adding wheel. This design of this base is an adaptation of the uBot-5 platform developed at the University of Massachusetts at Amherst [58]. The 2 degrees-of-freedom hands and the 17 degrees-of-freedom face are unused in the work reported here, except periodic blinking. In extensions of this work, we plan to use the robot's social expressiveness to communicate its intention to the trainer (more on this topic the "transparency" item in the future research list in Chapter 7).

In addition to the Vicon system described above, the robot has a number of other sensing capabilities. These include a laser-range finder mounted at its torso, a color camera at each eye, a 3D infrared depth-sensing camera embedded in its forehead, and a four-microphone array in its head. Additionally, each joint is monitored by high-resolution encoders and current sensing.

The MDS platform supports on-board and off-board computation. On-board computation is responsible for sensor management, low-level control, and wireless communication, whereas off-board computation performs higher-level processes of perception, cognition, learning, and behavior. The MDS robot Nexi runs on the *r1d1* cognitive architecture, which we integrated with TAMER. Note that this research does not use the full capabilities of Nexi; rather, we largely restrict our use of the sensing and acting functions to those that are necessary to teach interactive navigational tasks.

TAMER algorithm for interactive robot navigation

We implemented the full TAMER algorithm as described in Algorithm 5, assigning credit by the current method of delay-weighted aggregate reward (see Section 3.3), with the following components.

- The *representation of* $\hat{R}_H$ and the *updateModel() method* are both subsumed by the k-nearest neighbors algorithm. In addition to the action, the two features input to $\hat{R}_H$ are the distance and angle to the human from the robot's perspective. The passage below gives more detail.
- f_{delay} is a Uniform(-0.8 seconds, -0.2 seconds) distribution.
- Since f_{delay} does not have a long tail, ϵ_p is simply 1.
- c_{min}, which controls extrapolation from unfinished samples, is 0.5.
- The *training interface* is different from that described in Section 3.4.2 in two ways. First, the trainer's visual perception of the robot and its environment clearly differs from perception of a simulated environment. Second, the positive and negative reward buttons—which still map to $+1$ and -1—are on a presentation remote that can be held in the trainer's hand. Also, an additional button on the remote toggles the training session on and off (see Section 3.3.6 for details of such toggling). Another button both turns training off and forces the robot to stay still. This safety function is meant to be used when the robot appears likely to collide with an object in the environment.

From the robot's estimation of the position and orientation of itself and the training artifact, two features are extracted and used as input to $\hat{R}_H$. The first feature is the distance in meters from the robot to the training artifact, and the second is the angle in radians from the robot's position and orientation to the artifact.

$\hat{R}_H$ is modeled using k-nearest neighbor with a separate sub-model per action (i.e., there is no generalization between actions). The number of neighbors k is dynamically set to the square root of the number of samples gathered for the corresponding action, rounded down to the nearest integer. The distance metric is the Euclidean distance given the 2-dimensional feature vectors of the queried state and the neighbor's state.

To help prevent a single highly negative reward during early learning from making Nexi avoid the targeted action completely, we bias $\hat{R}_H$ toward values of zero. This biasing is achieved by reducing the value of each neighbor by a factor determined by its distance d from the queried state. The bias factor is calculated as the maximum of linear and hyperbolic decay functions: $max(1-(d/2), 1/(1+5d))$, where d is the same Euclidean distance metric used to choose nearest neighbors.

Models with zero samples output predictions of 0. When multiple actions have the same predicted reward from the current state and one of those actions was taken in the previous step, that same action is taken again. If the previous action is not part of the tie, one of the tied actions is chosen randomly. Accordingly, at the first time step, during which all actions are tied with a value of 0, a random action is chosen.

Results and discussion

I now describe the results of training Nexi and discuss challenges and lessons provided by implementing TAMER in this domain. Five different behaviors were successfully taught by the author:

- **Go to** – The robot goes to the artifact and stops before it with little space between the two.

- **Keep conversational distance** – The robot goes to the artifact and stops at an approximate distance from the training artifact that two people would typically keep between each other during conversation (about 2 feet).

- **Look away** – The robot should turn away from the artifact, stopping when facing the opposite direction. The robot never moves forward.
- **Toy tantrum** – When the artifact is near the front of the robot, it does not move, as if the artifact is a toy that is in the robot's possession. Otherwise, the robot turns from side to side, as if in a tantrum to get the toy back. The robot never moves forward.
- **Magnetic control** – When the artifact is behind the robot, it acts as if the artifact repels it. The repulsion is akin to one end of a magnet repelling another magnet that faces it with the same pole. Specifically, when the artifact is near the center of the robot's back, the robot moves forward. If the artifact is behind its left shoulder, it turns right, moving that shoulder forward. And vice-versa for the right shoulder. If the artifact is not near the robot's back, the robot does not move.

Figure 3.15 provides an iconic illustration of each behavior. Videos of the successful training sessions—as well as some earlier, unsuccessful sessions—can be seen at `http://www.cs.utexas.edu/~bradknox/videos/nexi`. All of the videos were taken during a one-day period of training and refinement of the implementation of the TAMER algorithm, where we specifically adjusted action durations, the effects of chosen actions, and the communication of the robot's perceptions to the trainer.

Nearly all sessions that ended unsuccessfully failed because of issues of *transparency*, which the author addressed before or during this period. By "issues of transparency", I mean mismatches between the state-action pair currently occurring and what the trainer believes to be occurring. The specific points of confusion and their solutions are described below.

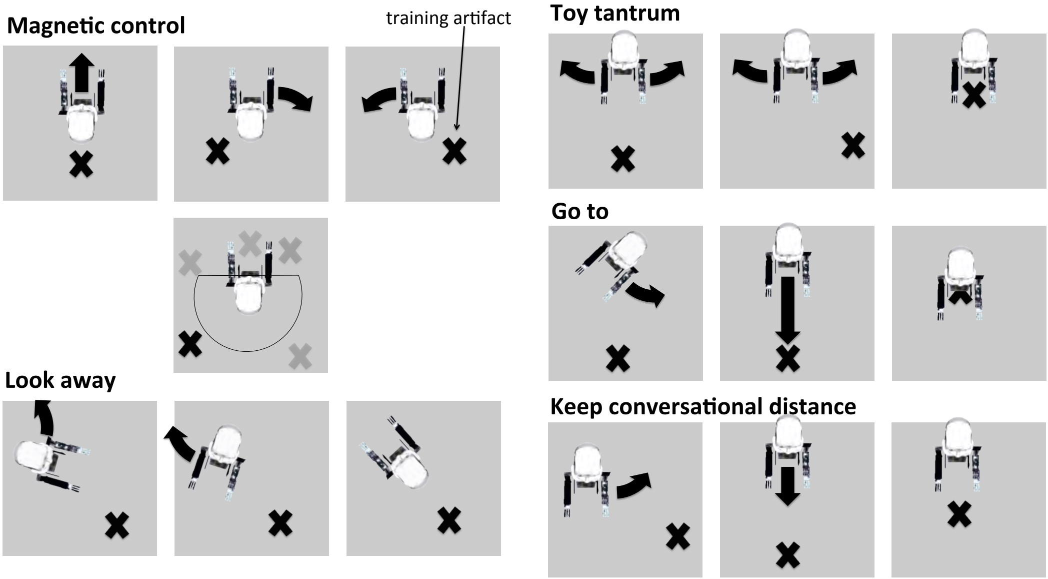


Figure 3.15: An iconic illustration of the five interactive navigational behaviors that were taught to the MDS robot Nexi. Each gray square represents a category of state space. The arrow indicates the desired action in such state; lack of an arrow corresponds to the stay action.

- *the start and end of actions* – As mentioned previously, there can be a delay between the robot taking an action (e.g., turn right at 0.15 rad/s) and the robot visibly performing that action. This delay occurs specifically after any change in action. This offset between the robot's and the trainer's understandings of a time step (i.e., the duration of an action) can cause reward to be misattributed to the wrong action. The durations of each action were chosen to ensure that the robot will carry out an action long enough that its visible duration can be targeted by the trainer.

- *the action being carried out* – Though this point of confusion overlaps with the previous one, its cause is different. TAMER was interfaced with a complex code base meant to give wide and varied functionality to Nexi. One such function calculated and executed a trajectory to a given waypoint. In early implementations, waypoints were used to execute the abstracted atomic actions that TAMER chooses from. However, in rare cases, the robot would calculate strange trajectories, such as turning a full circle rightward to move a small angle leftward. During training, such trajectories suggested that the robot was taking a different action than it was. To solve this, the default navigation system was overwritten to make the robot simply try to move at a constant rotational or forward velocity, without planning to reach waypoints.

- *the state of the training artifact* – The position of the artifact, unlike that of the robot, was estimated from only Vicon data. When the artifact moved beyond the range of the infrared Vicon cameras, its position was no longer updated. The most common source of failed training sessions was a lack of awareness by the trainer of this loss of sensing. An audible alarm was added that fired whenever the artifact could not be located, alerting the trainer that the robot's belief about the artifact is no longer changing.

Though the transparency issues above are specific to the robotic platform used, we nonetheless suspect that they are illustrative of the types of challenges that are likely to occur with physically embodied agents.

The *go to* behavior was taught successfully early on, after which the author attempted to train the *magnetic control* behavior. When the *go to* behavior was learnt, moving the artifact beyond sensory range caused no problems; putting the artifact in the middle of the training area would make the robot move there. However, with the *magnetic control*, the robot was being "pushed". Consequently, once the artifact was out of range, the robot was even farther outside the training area, closer to possible collisions. And the artifact had to be close to the robot to control it. In these cases, the artifact could not be used to bring the robot back. Worse, the trainer did not know the artifact was not being sensed, so the artifact was often not brought back in range, and the robot continued to falsely believe that the artifact was relatively near it. Many sessions were aborted in this manner, almost always with confusion about why the robot had apparently been learning well and then suddenly starting acting unpredictably. After the out-of-range alarm was added, the remaining four behaviors were taught successfully in consecutive training sessions.

Though the robotic aspect of this task brought unique challenges, the within-task interactivity—the presence of trainer-controllable features—created at least two insights. First, interactivity empowered the trainer by facilitating *capturing* of behaviors. In the animal learning literature, a trainer captures behavior by eliciting it or simply waiting for it and then giving reward [79, 12]. Captured behavior gives an initial approximation of the desired behavior that the trainer can then shape. The robot trainer's initial strategy was to watch what the robot first does. If the initial action was correct for the current artifact position, then positive reward was given. If it was incorrect, negative reward was given until the correct action was chosen. When a previously "punished" action was later desired, the prediction of human reward was so low that the robot wouldn't try the action until all other actions were predicted to receive even *lower* rewards. This race to the bottom would quickly reach a point where the trainer could not elicit an action. We partially addressed this issue through the distance-based biasing described at the beginning of Section 3.4.4, which reduces the strength of generalization in $\hat{R}_H$; however, a number of negative rewards early during training could still be problematic. A breakthrough occurred when the trainer began moving the artifact to a location such that the initial action was correct and then giving *positive reward*. The feature space for that action was "painted" with positive rewards while moving the artifact to all areas of the feature space where the action was desired. Only after this painting—giving a strong positive bias to predictions of reward for that action in any state–was negative reward given to change the action, after which the new action was likewise painted. This positive-reward painting strategy is a form of capturing, and it is only possible with interactive features. I summarize this strategy as follows: *if the robot misbehaves during early training, try to change the context so that the misbehavior is correct and then give positive reward.*

The second insight is based upon the *magnetic control* behavior. Here, human reward is used to train a previously unknown control interface. This control interface could then be used for further learning. For example, the robot could be controlled to give demonstrations for imitation learning. With trainer-controllable features, initial training could form the lower level of a learning hierarchy, in which relatively simple behaviors are trained and then composed together to perform a more complex behavior.

3.5 Summary

At a high level, the TAMER framework breaks the process of interactively learning tasks from human reward into 3 modules: assigning credit from delayed reward to recent behavior, supervised learning of a model of human reward, and selecting actions based on the human reward model. In contrast to agents that learn from the output of an MDP's reward function, TAMER works in the absence of a predefined evaluation metric (i.e., reward function), can reach good performance with relatively few samples, and facilitates teaching new behaviors by almost any user, regardless of his or her programming skills. And unlike previously deployed algorithms that learn from human-delivered reward, TAMER treats human reward differently than MDP reward. Our approach is argued intuitively in Section 3.1, shown to be effective for teaching agents to perform multiple tasks in Section 3.4, and given empirical backing in comparison to other existing approaches by our analysis in Chapter 6 (which ultimately opens the door to mapping human reward to a subset of the various types of MDP reward).

In comparison to algorithms that learn from MDP reward, our experimental data suggests that TAMER can greatly reduce the samples required to learn a good policy. The data also suggests that well-tuned agents learning from MDP reward are generally better at maximizing final, peak performance after many more trials. In the following chapter, we explore how to learn from both human and MDP reward (when available), leveraging their complementary strengths. We test eight different techniques for combining a model of human reward, $\hat{R}_H$, with a reinforcement learning algorithm. The two most successful techniques, both of which only affect action selection, achieve better results than either TAMER or the reinforcement learning algorithms can achieve in isolation.

This chapter focuses primarily on the TAMER algorithm. Insights about human trainers motivate TAMER, but we do not consider how a trainer's feedback is affected by decisions in the design of the agent, the training interface, or the instructions to the trainer. Chapter 5 describes two experiments that examine such manipulations of the trainer.

In Chapter 6, we explore the crucial question of how predicted values of future human reward should be discounted, which TAMER answers differently from past work. Relatedly, I discuss what may possibly the most important advantage of interactive shaping over other many forms of teaching: the agent can improve upon the policy that the trainer intends to teach. Such successful outcomes would occur if a human trainer, through reward, could teach the goals of a task rather than a policy. Unfortunately, TAMER itself does have the ability to improve its policy by planning; TAMER's myopic action selection does not attempt to maximize its accumulation of human reward. In Chapter 6, we justify TAMER's myopic behavior by identifying a problem with seeking maximum long-term human reward (the "positive circuits" problem). However, we then find a method for modifying the task to avoid this problem, pointing forward to algorithms for interactive shaping that build upon yet are even more powerful than TAMER.

Chapter 4

TAMER+RL

Chapter 3 introduces TAMER *and compares interactive shaping to algorithms that learn from a reward function that is pre-programmed as part of the task's Markov decision process (MDP) specification. This comparison shows that interactive shaping can greatly reduce the samples required to learn a good policy. However,* TAMER *is designed to learn only from human reward; when MDP reward is also available, both feedback signals should be used. This chapter examines how to learn from both human and MDP reward, leveraging the fast learning exhibited by the* TAMER *framework to hasten a reinforcement learning (RL) algorithm's climb up the learning curve.*[1] *In other words, whereas Chapter 3 focused on a learning scenario in which the human teaches new behavior for a task that might have previously been undefined, this chapter instead examines how to use human teaching to improve the learning of a predefined task.*

In a sequential setting, we test eight plausible TAMER+RL *methods for combining a previously learned human reward function,* $\hat{R}_H$*, with MDP reward in a reinforcement learning algorithm. Results for two of these sequential* TAMER+RL *algorithms consistently demonstrate better final performance and better cumulative*

[1] This chapter is built from previously published work by the author [48, 52].

performance than either a TAMER *agent or an RL agent alone.*

We then adapt the most successful methods to learn simultaneously from both reward sources, enabling human feedback to be incorporated at any time during the reinforcement learning process. We call these algorithms simultaneous TAMER+RL*. To enable simultaneous learning, I introduce a technique that appropriately determines the magnitude of the human model's influence on the RL algorithm throughout time and state-action space.*

The TAMER framework guides the design of agents that learn by shaping—using signals of approval and disapproval to teach an agent a desired behavior. In Chapter 3, I describe TAMER fully. As originally formulated, TAMER was limited to learn exclusively from the human feedback. Experimental analysis of TAMER shows that shaping can greatly reduce the number of samples required to learn a good policy. Comparisons with autonomous learning agents also suggest that human reward—directly evaluative yet flawed—and reward from a Markov Decision Process (MDP)—indirectly evaluative yet containing a flawless specification of behavior—are complementary signals that could be used together when a reward function is available.

As designed, TAMER does not permit human training to be combined with learning from an MDP reward function. When the human trainer should be the determinant of correct behavior, TAMER is well suited for learning. However, for predefined tasks, a human user who already has significant task knowledge should fill a different role. We seek to enable non-technical users to transfer such knowledge to the agent through signals of human reward, reducing the cost of learning without hurting the agent's final, asymptotic performance. More specifically, when the task is predefined and a hard-coded reward function is available, an agent should ideally be able to learn from both human and MDP reward. This chapter examines how to best combine interactive shaping via the TAMER framework with reinforcement learning (RL) from MDP reward.

In this chapter, we test eight plausible methods for combining a model of human reward—learned by TAMER—with MDP reward in a reinforcement learning algorithm, SARSA(λ) [90]. After some motivating background material in Section 4.1, Section 4.2 describes the eight combination techniques.

In Section 4.3, we examine the case where learning from human reward occurs only prior to RL (i.e., learning from MDP reward): *sequential* TAMER+RL. Since human interaction occurs before the combination of the human reward model with RL, the specific combination technique used has no effect on the human trainer. Therefore, this sequential learning scenario allows deep analysis of different TAMER+RL algorithms without requiring distinct human data for each one. We deem a TAMER+RL technique successful if, after a large, predetermined number of runs, either the cumulative MDP reward received by the learning agent or its final performance level is greater than it would be if using the RL algorithm alone or if greedily exploiting the human reward model, as a TAMER agent would. By these criteria for success, several methods achieve positive results. I discuss which of these methods are most effective and analyze why some methods work and others do not help. Additionally, we evaluate the successful techniques' performances at a range of parameter values to determine the ease of setting parameters effectively, a critical aspect of using TAMER+RL algorithms in practice.

We remove this sequential constraint in Section 4.4, developing and testing TAMER+RL algorithms that learn *simultaneously* from both human and MDP reward. The principal benefit of simultaneous learning is its flexibility; it gives a trainer the important ability to step in as desired to alter the course of reinforcement learning while it is in progress. We demonstrate the success of extensions of the two best-performing techniques from our sequential experiments, action biasing and control sharing, in this simultaneous setting. To meet demands introduced by the simultaneous setting, we develop a method to moderate the influence of the model of human reward on the RL algorithm. Using this method, simultaneous TAMER+RL increases the human model's influence in areas of the state-action space that have recently received training and slowly decreases influence in the absence of training, leaving the original MDP reward and base RL agent to learn autonomously in the limit. Without this improvement, the sequential techniques would be too brittle for simultaneous learning.

4.1 Reinforcement learning and TAMER

In this section, I describe the reinforcement learning component the TAMER+RL algorithms and then motivate the chapter through comparison of the MDP reward signal typically used in RL to a human reward signal.

4.1.1 Markov decision processes and reinforcement learning

In Chapter 2, I gave an introduction to Markov decision processes (MDPs) and reinforcement learning. In this chapter, I focus on augmenting value-function-based RL methods with TAMER-based learning from a human's reward signal. To represent value-function-based methods, our experiments use the RL algorithm SARSA(λ) [90] to learn the action-value function (i.e., a Q-function), for reasons discussed in Section 4.3.1.

4.1.2 The MDP reward signal

The reward signal within an MDP is often characterized as *sparse* and *delayed*. To illustrate, consider a Markov Decision Process that describes a chess game. Within a typical formulation, the reward function would yield zero reward for any state-action pair that does not terminate the game. A state-action pair that transitions to a win might receive a $+1$ reward, a loss would receive a -1 reward, and a stalemate would receive a 0 reward. This reward signal is sparse because discriminating reward is rarely received. It is delayed because any non-terminating state-action pair, regardless of its quality, receives 0 reward, and so the agent must wait until the end of the game to receive any information from the environment that helps it determine the quality of that state-action pair. Further, even immediate, discriminating reward must be considered as only one component of an evaluation that also includes predictions of discounted future reward from paths down an exponentially branching tree of possible state-action pairs. These challenges represent the credit assignment problem of reinforcement learning; MDP reward informs the value of all previously experienced state action pairs. In short, MDP reward only *indirectly evaluates behavior.*

However, MDP reward can be described as *flawless in its specification of behavior*; along with the transition function, MDP reward determines optimal behavior—the set of optimal polices that, for each state, choose the action with the highest possible return.

4.1.3 The human reward signal

In this chapter, we combine a model of human reward ($\hat{R}_H : S \times A \rightarrow \mathbb{R}$) learned by TAMER with an RL algorithm learning from MDP reward. The TAMER framework is described fully in Chapter 3 and summarized as a reader's aid in Appendix A. TAMER circumvents the sparse and delayed nature of the MDP reward signal by replacing it with a human reward signal.

Under the assumptions of TAMER, a human reward signal is *directly evaluative*, containing information about whether the targeted behavior is good or bad in the long term. Unlike MDP reward, human reward is not sparse in its discrimination[2]—each reward fully distinguishes between approved and disapproved behavior—and it can be delivered with trivial delay, as subjects demonstrate in our user studies of TAMER agents (Section 3.4.1 and when $\gamma = 0$ in Section 6.4).

Note though that human reward is, in general, fundamentally *flawed*. Humans make mistakes, often have low standards, get bored, and have many other imperfections, so their evaluations will likewise be imperfect.

Though flawed, human reward can be exploited to more efficiently learn good, if not optimal, behavior, as I report in Chapter 3. In Section 3.4.2, we compare implemented TAMER agents to reinforcement learning agents in two contrasting domains: Tetris and mountain car. In short, we find that the interactively shaped agents strongly outperform RL agents in early training sessions, quickly exhibiting qualitatively good behavior. As the number of training episodes increases, however, many of the autonomous agents surpass the performance of TAMER agents. This chapter aims to combine TAMER's strong early learning with the often superior long-term learning of autonomous agents.

4.1.4 TAMER+RL

In Sections 4.1.2 and 4.1.3, I argued that MDP reward only indirectly evaluates behavior yet specifies optimal action and that human reward directly evaluates behavior but is flawed. This observation fits the aforementioned experimental results well. With more direct feedback, TAMER agents were able to learn much more quickly. But the signal was flawed and TAMER agents, in some cases, plateaued at lower performance levels than autonomous learning agents.

The TAMER framework does not use MDP reward. This characteristic can be a strength. It allows lay users to fully determine the goal behavior without defining and programming a reward function. However, it can also be a weakness. When

[2] However, human reward can be sparsely delivered.

the goal behavior is previously agreed upon and a reward function is available, a TAMER agent is ignoring valuable information in the reward signal—information which complements that found in the human reward signal.

In this chapter, we ask how one could use the knowledge of a previously trained TAMER agent to aid the learning of a RL agent. For all TAMER+RL approaches, only MDP reward is considered to specify optimal behavior. $\hat{R}_H$ provides guidance but not an objective, contrasting with the TAMER-only approach (described in Chapter 3) that addresses the Interactive Shaping Problem. From previous results, we expect that the gains in performance will be most pronounced in early learning.

4.2 Candidate techniques for combining TAMER and RL

Here I propose and specify eight techniques for combining TAMER and subsequent RL. We assume that the RL algorithm learns an action-value function Q, where $Q(s, a)$ is an estimate of return from (s,a) under the current policy. Thus our experimental analysis will not apply to policy search algorithms.

One goal of this research is to create tools that allow human knowledge to be added to reinforcement learning algorithms. For these tools to be practical, they should be generally and easily applicable to any RL algorithm that learns a Q-function. With that in mind, we placed the following restrictions on our methods for combining TAMER with RL:

1. The combination techniques should be independent of the model representations used for $\hat{R}_H$ and Q.

2. The influence of $\hat{R}_H$ needs to recede with time or with repeated visits to the same or similar states to keep the set of optimal policies unchanged.

3. We restrict the RL parameters to those which were found (through parameter tuning, described later) to yield good cumulative performance when the RL algorithm runs without any human influence. We do not tune parameters of the RL algorithm for each technique, but rather keep them constant across all techniques.

I list below the eight TAMER+RL methods that we tested for combining TAMER with subsequent reinforcement learning. Symbols with a prime character (e.g., Q') signify that they are replacing the already existing, corresponding non-prime symbols in either the MDP specification or in the agent's feature vector or Q-function. The parameter β moderates the magnitude of $\hat{R}_H$'s influence on the pure RL algorithm. When not otherwise stated, β has a constant value.

1. $R'(s,a) = R(s,a) + (\beta * \hat{R}_H(s,a))$. Here, the MDP reward is replaced with the sum of itself and the weighted prediction of human reward. The initial weight β is an algorithmic parameter and is decayed by a constant factor at some recurring point. The weight β must decrease in the limit to avoid changing the set of optimal policies.
2. $\overrightarrow{\phi}'_{s,a} = \overrightarrow{\phi}_{s,a}.append(\hat{R}_H(s,a))$. This technique assumes that the Q-function is learned over a feature vector that is drawn from some state-action pair. To the original feature vector $\overrightarrow{\phi}_{s,a}$, it adds $\hat{R}_H(s,a)$ as one additional feature, aiming to embed the knowledge contained in $\hat{R}_H(s,a)$ in the feature vector such that the RL algorithm can use it as much or as little as is needed. For linear models over the features, to bias initial action towards choosing as the TAMER agent would in early learning, this technique has an input parameter that sets the initial weights corresponding to the added feature (one weight per action).[3]

[3] This reliance on a linear function is our only violation of the three restrictions above (violating the first restriction). If the input parameter is zero, it does not violate the restriction.

3. *Initially train* $Q(\cdot,\cdot)$ *to approximate* $(\beta * \hat{R}_H(\cdot,\cdot))$. Treat $\hat{R}_H$ as the Q function. For our experiments, we randomly created 100,000 state-action pair samples for the Q function to train on.

4. $Q'(s,a) = Q(s,a) + [\beta * \hat{R}_H(s,a)]$. This method adds a weighted prediction of human reward to $Q(s,a)$ for any occasion that it is used, including for action selection and for updating Q (e.g., while calculating temporal-difference error).

5. $A' = A \cup argmax_a[\hat{R}_H(s,a)]$. To the set of possible actions, this method adds the action that the greedy TAMER agent would choose. An input parameter sets the Q value of the new action uniformly across states.

6. $Q'(s,a) = Q(s,a) + [\beta * \hat{R}_H(s,a)]$ *only during action selection.* This technique differs from the fifth technique by not directly affecting updates to the Q function. The weight β decreases in the limit to allow unbiased control by the RL algorithm.

7. $P(a = argmax_a[\hat{R}_H(s,a)]) = min(\beta, 1)$. *Otherwise original RL agent's action selection mechanism is used.* This method effectively either lets the RL agent choose its action normally or has the TAMER agent choose while the RL agent observes and updates as if it were making the choice. The action is chosen by $\hat{R}_H$ with probability $min(\beta, 1)$, where β decreases in the limit to allow full control by the RL algorithm.

8. $R'(s_t, a) = R(s,a) + \beta * (\phi(s_t) - \phi(s_{t-1}), where\ \phi(s) = max_a \hat{R}_H(s,a)$. This technique converts $\hat{R}_H$ into a potential function ϕ to determine supplemental reward.

Note that the Method 1 treats $\hat{R}_H(s,a)$ as MDP reward or a component of it, and Method 8 similarly uses $\hat{R}_H$ to change the reward signal, but only uses state-specific (and not action-specific) information from $\hat{R}_H$. Method 2 seeks to add the

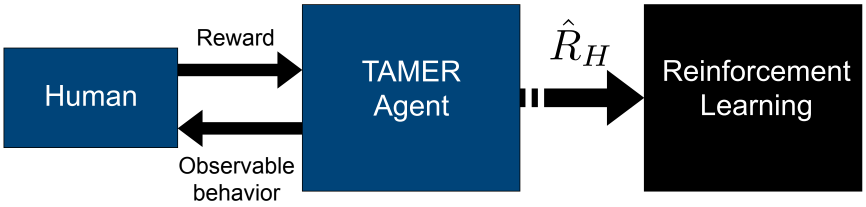


Figure 4.1: A conceptual diagram of sequential TAMER+RL. The human interacts directly with the TAMER agent. After training completes, the model of human reward, $\hat{R}_H$, is used by the RL algorithm along with MDP reward. Note that the task environment and MDP reward are not explicitly shown here; both the TAMER agent and the augmented RL agent interact with the environment, but only the RL agent receives MDP reward.

information contained in $\hat{R}_H$. Methods 3 and 4 interpret human reward as estimates of expected return from (s, a). Method 5 simply gives an extra action and biases the learner towards choosing that action early on. Methods 6 and 7 bias action selection towards exploiting $\hat{R}_H$ without affecting the updates to Q. These two approaches can be loosely interpreted as using $\hat{R}_H$ to perform demonstrations for the SARSA(λ) algorithm, creating something akin to imitation learning [4].

4.3 Sequential TAMER+RL

For Section 4.3, I focus on the scenario in which a human trainer has already trained a TAMER agent, and the learned human reward function, $\hat{R}_H$, is available to guide a reinforcement learning agent (illustrated in Figure 4.1). As I explain in this chapter's introduction, the sequential scenario has at least one crucial benefit: human data is independent of the combination technique and the parameters of the RL algorithm, allowing a thorough exploration of different TAMER+RL algorithms without incurring the cost in time and added variance of repeating training for each condition of our experiments.

We conduct two sets of sequential TAMER+RL experiments. The first set examines all eight combination methods and both optimistic and pessimistic initialization of the SARSA(λ) agent's Q function. Sections 4.3.1–4.3.3 describe this first set. Then, in the second set of experiments, we conduct a deeper analysis of those combination techniques that were found successful in the first set. This analysis includes multiple task domains and a report of each technique's sensitivity to its combination parameter β. This further analysis is described in Sections 4.3.4–4.3.6.

4.3.1 Sequential TAMER+RL experiments

To test the eight techniques for combining TAMER and reinforcement learning, we use SARSA(λ) as our reinforcement learning algorithm and test it within the mountain car domain. Recall from Section 3.4.1 that mountain car involves a simulated car, starting at a random location near the bottom of two adjacent hills, trying to get up one hill to a destination point on top. To gain enough momentum to climb the hill, the car must choose acceleration to go back and forth on the hills. To approximate $Q(s, a)$, the SARSA(λ) algorithm maintains a linear model over features generated by three (one for each action) two-dimensional grids of Gaussian radial basis functions (RBFs). At each time step, the linear function approximator updates via gradient descent.

SARSA(λ) with Gaussian RBFs is known to perform well in mountain car [90], though more effective algorithms do exist. However, our goal for the completed research on TAMER+RL is not to study the interaction between TAMER and different RL algorithms but rather to establish that they can be combined effectively and to study the different ways for doing transfer. We have no reason to expect qualitatively different results with other value-function based RL algorithms, but we leave an investigation of possible differences as future work.

We use two differently parametrized versions of SARSA(λ), each of which has six tuned parameters used by SARSA(λ) or for generating state-action features. I will refer to the two parameter sets as the optimistic set and the pessimistic set. For one set, state-action values are optimistically initialized at 0. For the other set, state-action values are pessimistically initialized, starting at approximately -120, based on our observation that a good policy can reach the goal in roughly 120 seconds or less from any state. Each set was tuned[4] to receive the highest amount of cumulative reward over 500 learning episodes (i.e., to take the least total time getting to the goal 500 times).

For these experiments, we used the TAMER algorithm described in Chapter 3, learning from the training data first described in Section 3.4.2. The algorithm represents $\hat{R}_H$ as a linear model and updates $\hat{R}_H$ by incremental gradient descent. For each action, a set of two-dimensional Gaussian radial basis functions over state are calculated as described by Sutton and Barto [90] and input to $\hat{R}_H$. These TAMER agents employ the earlier individual reward crediting technique described in Chapter 3. For all TAMER algorithms described this chapter, the delay probability distribution f_{delay} is Uniform(0.2 seconds, 0.8 seconds).

Training sessions by two trainers provide two static, previously learned human reward functions $\hat{R}_H$, each outputting approximately within the range [-1.0, 1.0]. One session was chosen because it yielded a function, which I will call $\hat{H}_1$, that performed near the middle—in terms of total MDP reward—of the functions created

[4] Parameters were tuned via a hill climbing algorithm that selected one of the features, tested out many different values for that feature, and then repeated. Tuning was stopped when there was no noticeable improvement over a few iterations. The six tuned parameters are the internal discount factor γ; the step size α for updates; λ for eligibility traces; ϵ for ϵ-greedy action selection; the width σ^2 of the Gaussian RBFs, following Sutton and Barto's definition of an RBF's "width" [90] and where the unit is the distance in normalized state space between adjacent Gaussian means; and the number of RBF features j. The optimistic parameter set is $\{\gamma = 1, \alpha = 0.08, \lambda = 0.87, \epsilon = 0.003125, \sigma^2 = 0.163125, j = 4800\}$, and the pessimistic parameter set is $\{\gamma = 1, \alpha = 0.15, \lambda = 0.84, \epsilon = 0.00625, \sigma^2 = 0.08, j = 4800\}$. For each action, the means of the 2-dimensional RBFs are placed along a 40×40 grid in state space. There were also some hand-tuned parameters: In addition to its RBF features, each action has an additional, constant feature of value 0.1. For each action, this feature and its RBF features are 0 when another action is input to Q. SARSA(λ) uses replacing traces, ϵ is annealed after each episode by a factor of 0.99, and the weights of the linear model of Q are each initialized to 0 in the optimistic case and -10 in the pessimistic case.

by the 19 participants from the mountain-car experiment described in Chapter 3, making it representative of a typical trained TAMER agent. The other was chosen because it yielded the best function (called $\hat{H}_2$) of the 19.

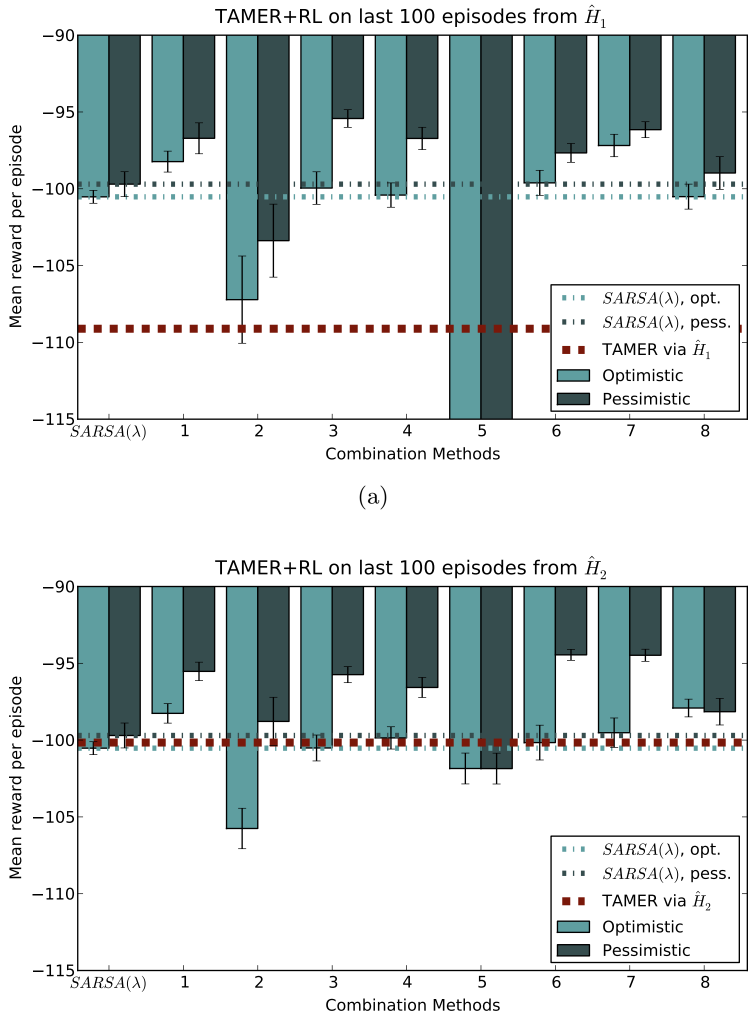
Recall that three combination techniques—1, 6, and 7—eventually decrease their combination parameter β so that, in the limit, the algorithm is effectively SARSA(λ) acting on and learning from the original MDP. For these sequential TAMER+RL experiments on all eight techniques, this decrease is achieved by annealing β by a factor of 0.98 at the end of each episode. Also, for the same experiments, $\beta = 1$ for technique 7 at start, initially giving full control to $\hat{R}_H$.

For each technique that requires an input combination parameter (expressed as β for some techniques), we test four to six different parameters, all powers of ten. In the following section, I report the results using the best parameter for each technique, as determined by mean cumulative reward.

4.3.2 Sequential TAMER+RL results

Success of a TAMER+RL combination technique can be defined by four criteria. Two such criteria are to achieve a higher *final performance* than either SARSA(λ) alone—which I term "SARSA(λ)-only" for clarity—or by exploiting the static, previously learned $\hat{R}_H$ directly—which I term "TAMER-only". The third and fourth criteria are to receive more reward over 500 episodes—*cumulative performance*—than either SARSA(λ)-only or TAMER-only. Since SARSA(λ)-only performs best with optimistic initialization, we will compare the combination methods to optimistic SARSA(λ)-only exclusively.

Final performance To evaluate the final performance level achieved under each combination, we look at the mean reward received over the last 100 episodes, shown in Figure 4.2. (Looking at many episodes removes some of the effects of the stochas-



(a)

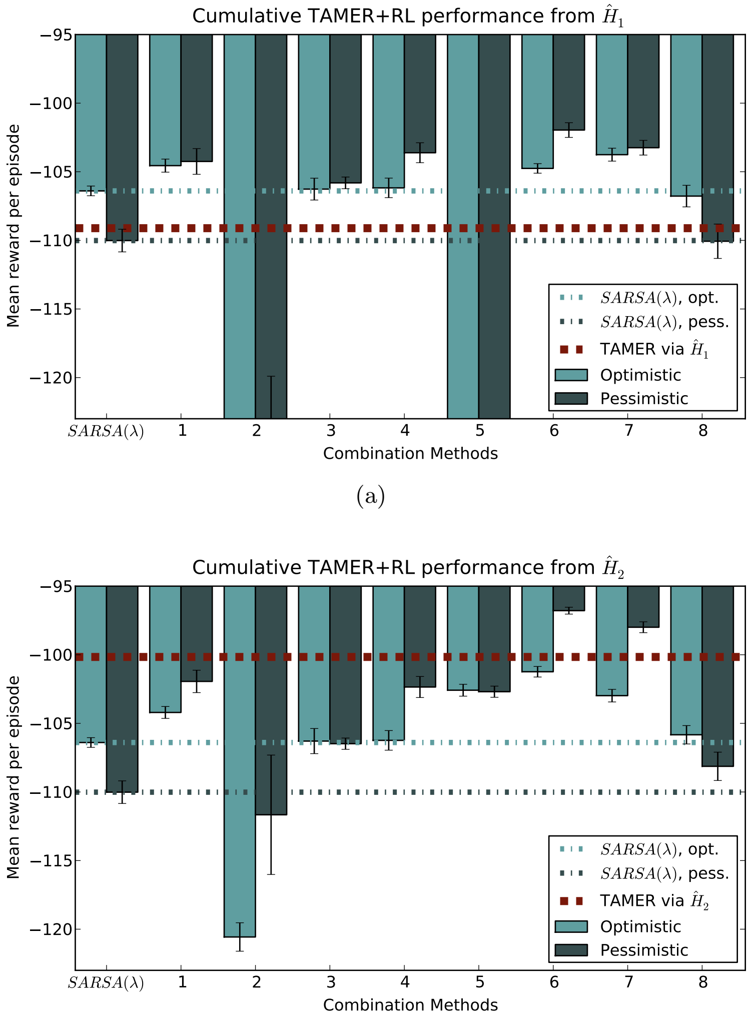
(b)

Figure 4.2: The mean reward received per episode by each of the eight combination techniques over the last 100 episodes of 30 or more runs of 500 episodes. Each bar graph describes experiments using different trained $\hat{R}_H$ functions. The two $\hat{R}_H$ functions display the effect of two different levels training quality. The performance of SARSA(λ)-only under optimistic and pessimistic initialization and the mean reward received by the static TAMER-only policy exploiting $\hat{R}_H$ are given for comparison. Error bars show 95% confidence intervals, assuming that mean reward over a run is normally distributed.

tic initial start state for each episode.) The combination techniques are generally quite effective at improving the final performance of SARSA(λ). With pessimistic initialization, all of the techniques improve performance except Methods 2 and 5, which are the worst two methods with respect to cumulative reward as well. Also, Method 8 only marginally improves performance with one $\hat{R}_H$. However, under optimistic initialization, the combination techniques' final performance improves on that of SARSA(λ)-only less often and to a lesser extent than the corresponding pessimistic techniques. Under optimistic initialization, only Method 1 consistently has better final performance than SARSA(λ)-only. Consequently, I focus my subsequent writing on pessimistic initialization; the reader should assume that I speak of the pessimistic set when I do not explicitly state which initialization method I am discussing.

Cumulative reward From Figure 4.3, we can evaluate the second condition for success—the mean reward received across all 500 episodes during all runs. For the most part, each combination technique performs similarly with both $\hat{H}_1$ and $\hat{H}_2$. Nearly every technique performs better with pessimistic initialization than with optimistic initialization, which I discuss in Section 4.3.3. The best technique for both $\hat{R}_H$s is pessimistic Method 6, $Q'(s,a) = Q(s,a) + [\beta * \hat{R}_H(s,a)]$ *only during action selection*, followed by the similar Method 7, $P(a = argmax_a[\hat{R}_H(s,a)]) = min(\beta, 1)$. Methods 1, $R'(s,a) = R(s,a) + (\beta * \hat{R}_H(s,a))$; 6; and 7 are the only techniques which perform better under both initialization types than optimistic SARSA(λ)-only. Method 4, where $Q'(s,a) = Q(s,a) + [\beta * \hat{R}_H(s,a)]$, also outperforms optimistic SARSA(λ)-only across both $\hat{R}_H$s.

Method 3, training $Q(s,a)$ to approximate $(constant * \hat{R}_H(s,a))$, improves upon SARSA($\lambda$)-only a small amount under pessimistic initialization, but does not consistently outperform optimistic SARSA(λ)-only. Method 8 exhibits small im-

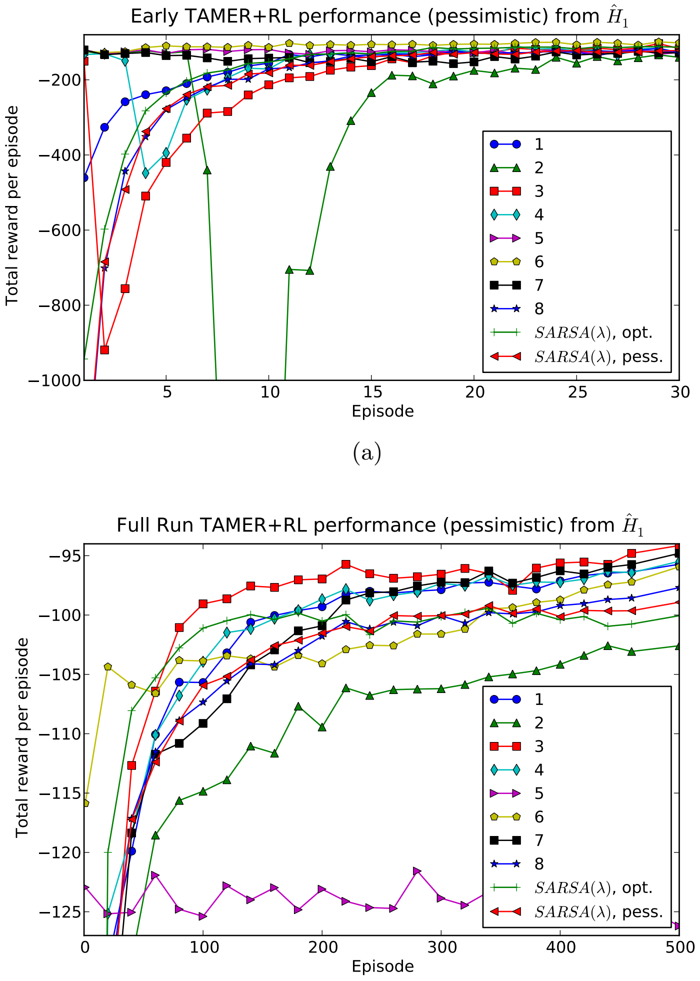


(a)

(b)

Figure 4.3: These bar graphs are identical to those in Figure 4.2, except that the statistics are calculated over all 500 episodes, allowing us to assess cumulative agent performance under each combination method. (Also note the difference in the range of the y-axes.)

provement under $\hat{H}_2$, but overall is the least influential method. Method 5 is the only one that differs greatly between $\hat{R}_H$s, showing significant improvement with $\hat{H}_2$ and much worse performance with $\hat{H}_1$. Method 2, adding $\hat{R}_H(s, a)$ as an additional



(a)

(b)

Figure 4.4: Learning curves using $\hat{H}_1$ and pessimistic initialization. The first plot focuses on the early learning trials. The plot that shows the full run has been smoothed for readability.

state-action feature, consistently performs worse than SARSA(λ)-only.

Figure 4.4 shows learning curves for each technique with $\hat{H}_1$ and pessimistic initialization. The two plots display performance early in the run and across the entire run.

4.3.3 Sequential TAMER+RL discussion

In this section I discuss which combination methods effectively transfer knowledge from $\hat{R}_H$ to SARSA(λ). I identify patterns among the results, give explanations for the failures of certain methods, and discuss the impact of using optimistic versus pessimistic initializations.

Comparing the combination techniques The relative performance of the TAMER+RL techniques is qualitatively similar across the two $\hat{R}_H$s. For the "typical" human reward function, $\hat{H}_1$, several of the pessimistic combination techniques improve both cumulative reward and final performance compared to running SARSA(λ)-only or merely acting as a static TAMER-only agent, satisfying all of our standards for success (Section 4.3.1). The same techniques also perform well for the "best" available human reward function, $\hat{H}_2$, outperforming SARSA(λ)-only for both success criteria and achieving better final performance than the static TAMER-only agent. However, only Methods 6 and 7 achieve better cumulative reward than the high performance TAMER-only agent.

To determine whether the SARSA(λ)-only agents would, if given more time, reach the final performance level of the successful methods, we ran 10 runs of SARSA(λ)-only over 1000 episodes. The optimistic and pessimistic initializations respectively average -102.809 and -98.842 reward per episode during the last 100 episodes (starting after the 900th). Almost all of the pessimistic combination methods that outperform SARSA(λ)-only in final performance in the 500 episodes experiments also outperform it when SARSA(λ)-only is given 1000 episodes. Therefore combining SARSA(λ) with TAMER actually achieves performance levels that are otherwise unreachable in twice the training time—and possibly ever.

From the results, we notice two patterns:

- *Initially manipulating the model of Q correlates with poor performance.* Three of the worst four combination methods—2, 3, and 5—each involve an initial manipulation of the learning model. Method 3 changes the initial parameters of the model. Methods 2 and 5, the two worst performing methods, each change the input space of the model. If this pattern is a factor in performance, a plausible explanation is that the SARSA(λ) parameters are effective for the model under which they were tuned but are not robust to changes to the model.

- *Gently pushing the behavior of the learning agent toward what the* TAMER-*only agent would do and removing the influence of $\hat{R}_H$ slowly and smoothly correlates with good performance.* Three of the four methods that outperform SARSA(λ)-only in both cumulative reward and final performance—Methods 1, 6, and 7—exert influence on the agent through a weight that decays exponentially. The worst four methods do not use a decaying weight. Further, the two methods that create a form of demonstration (affecting action choice and nothing else) differ in the subtlety of their influence. Method 6 increases the Q-value during action selection by the weighted prediction of human reward. Contrastingly, $\hat{R}_H$ has all-or-nothing influence in Method 7. Either the TAMER+RL agent chooses the action via $\hat{R}_H$ or the SARSA(λ) algorithm chooses based on the current Q-function. The gentler Method 6 achieves significantly better cumulative performance (though it performs roughly equivalently during the last 100 episodes).

Aside from the possible influence of the above patterns, there is another plausible reason for the consistent failure of Method 2 (adding $\hat{R}_H(s, a)$ as an extra state-action feature) across $\hat{R}_H$s, initialization types, and success criteria (i.e., final performance or cumulative reward). Temporal difference learning algorithms such as SARSA(λ) often learn poorly when using function approximators over features that generalize any learning across large areas of the state-action space [13]. Within the linear model we created over state-action features, changing the weight for the $\hat{R}_H$ feature during a single update affects *every* state-action pair.

The impotence of Method 8, which uses a potential function over states, $\phi(s)$ to augment reward, suggests that the human reward signal more or less only carries information about the relative desirability of actions given a state, rather than information about the relative value of states. It is possible, though, that this characteristic of human reward occurred because the human trainers adapted to the TAMER system, which would not use such state information. From different perspectives, both Chapters 5 and 6 investigate how algorithmic choices affect the feedback of the trainer.

Taking into account both final performance and cumulative reward, Method 6 appears to be most effective. Under both $\hat{R}_H$s, it yields the best cumulative reward and essentially ties for the best final performance.

Optimistic versus pessimistic initialization Despite SARSA(λ) alone performing best with optimistic initialization, once it is combined with these eight methods, SARSA(λ) almost uniformly performs best with pessimistic initialization. Upon examining the step-by-step changes in state-action values during the first few episodes for several optimistically initialized methods, we noticed that any method which biases early behavior towards a certain action (given the state) is actually making the other actions have relatively higher state-action values. This effect occurs because when state-action values are optimistically initialized, they can only go down. So by biasing early behavior, we make the desired behavior look less desirable (from a Q-value perspective). Further, the only way that the agent can learn that the undesired state-action pairs are not as good as their optimistic initial values is by choosing those actions (which are all those actions that the TAMER-only agent would not make!) and learning that they are overvalued.

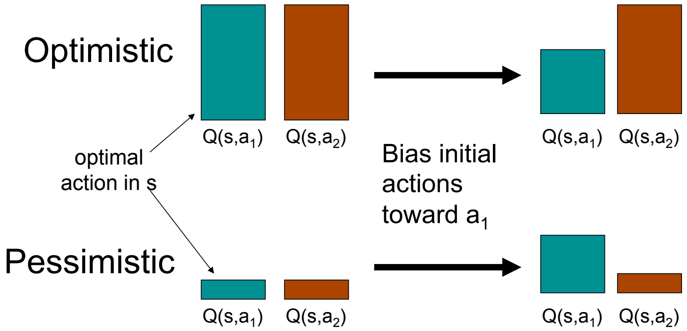


Figure 4.5: A simple illustration of the effects of initializing the Q-function optimistically or pessimistically. Here, initial actions are correctly biased towards the optimal action, a_1. With optimistic initialization, the $Q(s, a_1)$ decreases, making a_1 seem less desirable than suboptimal a_2—the opposite of our intended effect. Pessimistic initialization, on the other hand, yields the intended effect. The optimal action a_1 is made relatively more desirable by biasing early action selection toward it.

Observing that the techniques were not performing especially well under optimistic initialization, we instead tried pessimistic initialization. We set our model weights so that the initial state-action values were all approximately -120. Note that such an initialization is only pessimistic with regards to a good policy. It is actually optimistic for a low-performing policy. With such an initialization, biasing early actions towards good behavior increases the state-action values of the generally higher value state-action pairs that are being experienced while leaving pessimistic the state-action values of other, generally lower value state-action pairs (excepting the effects of function approximation). We believe that this initialization effect, illustrated in Figure 4.5, explains why the combination methods improve the RL algorithm more with pessimistic initialization.

4.3.4 Further analysis of the successful combination techniques: performance on two tasks and parameter sensitivity

Recent sections (4.3.1 to 4.3.3) introduce and test eight TAMER+RL techniques that each use $\hat{R}_H$ to affect the RL algorithm in a different way. From here to Section 4.3.6, we extend the work of previous sections by evaluating TAMER+RLtechniques on a second task, narrowing our focus to the most successful techniques, and analyzing the sensitivity of each technique to its combination parameter β. All of the experiments from this section onward use a reimplementation of our TAMER and TAMER+RL code, moving from a Python codebase to a Java one. This reimplementation is briefly discussed in Section 4.3.5.

Of the eight techniques, four are largely effective when compared to SARSA(λ)-only and TAMER-only agents[5] on both mean reward over a run and performance at the end of the run. We now focus on those four techniques, which we believe can be used on any RL algorithm that learns an action-value function and uses that function to select actions. Below, I list the techniques with names we create to facilitate deeper discussion. As before, in my notation β is a predefined combination parameter and a prime (e.g., Q') after a function means the function replaces its non-prime counterpart in the base RL algorithm.

- **Reward shaping**: $R'(s,a) = R(s,a) + (\beta * \hat{R}_H(s,a))$
- **Q augmentation**: $Q'(s,a) = Q(s,a) + (\beta * \hat{R}_H(s,a))$
- **Action biasing**: $Q'(s,a) = Q(s,a) + (\beta * \hat{R}_H(s,a))$ *only during action selection*
- **Control sharing**: $P(a{=}argmax_a[\hat{R}_H(s,a)]) {=}\ min(\beta, 1)$. *Otherwise use base RL agent's action selection mechanism.*

These four techniques are numbered 1, 4, 6, and 7 in Section 4.2. Now that

[5] In sequential TAMER+RL, $\hat{R}_H$ is constant, and thus so is the TAMER-only agent's policy.

we are narrowing our focus to the four most successful techniques, I briefly discuss each technique and situate it within related work.

I now briefly discuss these techniques and situate them within related work. In the RL literature, *reward shaping* adds the output of a shaping function to the original MDP reward, creating a new reward to learn from instead [24, 65]. As I confirm in the coming paragraph on Q augmentation, the reward shaping technique used here is not the only way to do reward shaping, though it is the most direct use of $\hat{R}_H$ for reward shaping.

If $\hat{R}_H$ is considered a heuristic function, action biasing is the same action selection method used in Bianchi et al.'s Heuristically Accelerated Q-Learning (HAQL) algorithm [6]. Control sharing is equivalent to Fernandez and Veloso's *π-reuse exploration strategy* [29]. Note that both control sharing and action biasing only affect action selection and can be interpreted as directly guiding exploration toward human-favored state-action pairs.

Q augmentation is action biasing with additional use of $\hat{R}_H$ during the Q-function's update. Wiewiora et al.'s related *look-ahead advice* [114] uses a discounted change in the output of a state-action potential function, $\gamma\phi(s_{t+1}, a_{t+1}) - \phi(s_t, a_t)$, for reward shaping and to augment action values during action selection. Interestingly, *look-ahead advice* is equivalent to Q augmentation when $\hat{R}_H$ is used for ϕ, the state and action space are finite, and the policy is invariant to adding a constant to all action values in the current state (e.g., policies derived from ϵ-greedy and softmax action selection).

A unified framing of reward shaping and action biasing Here, we provide a novel framing of reward shaping and action biasing that unifies them under one expression that differentiates between the two techniques by a single parameter.

The goal of SARSA(λ) and many other reinforcement learning algorithms is to learn the current policy's expected return, which for a state action pair (s_0, a_0) is $Q(s_0, a_0) = \sum_{t=0}^{T} E[\gamma^t R(s_t, a_t)]$. When shaping rewards by $\hat{R}_H$, this equation changes to $Q(s_0, a_0) = \sum_{t=0}^{t} E[\gamma^t (R(s_t, a_t) + \hat{R}_H(s_t, a_t))]$, which is equivalent to $Q(s_0, a_0) = \sum_{t=0}^{t} E[\gamma^t (R(s_t, a_t)] + \sum_{t=0}^{t} E[\gamma^t \hat{R}_H(s_t, a_t)]$ (we leave β out of the equations to increase readability). Therefore, learning from shaping rewards can be seen as learning a different Q function for each signal: $Q = Q_R + Q_H$.

For reward shaping, the discount factors of the returns for the two signals are equivalent. However, if the discount factor of Q_H, γ_H, is zero, then $Q_H(s_0, a_0) = \sum_{t=0}^{t} E[\gamma_H^t \hat{R}_H(s_t, a_t)] = \hat{R}_H(s_0, a_0)$. Therefore, $Q = Q_R + \hat{R}_H$. If $\hat{R}_H$ is known to the agent (as it is for a TAMER agent), then only Q_R needs to be learned and no part of its error will come from the known $\hat{R}_H$ component. Thus, greedy action selection with $\gamma_H = 0$ yields $Q(s, a) = Q_R(s, a) + \hat{R}_H(s, a)$, which is the action-biasing method.

Since MDPs typically have discount factors at or near one, reward shaping and action biasing can be seen as two extremes along a spectrum of possible γ_H values. Additionally, if both Q_R and Q_H specify the same set of optimal policies, then Q (where $Q = Q_R + Q_H$ still) also specifies that set of optimal policies. Intuitively, we want the agent to learn Q_H such that it encourages actions that are optimal for the MDP. Thus, the comparison between action biasing and reward shaping is a referendum on the assumptions that create Q_H. Action biasing follows the assumptions of TAMER, whereas reward shaping assumes that predictions of human reward can be learned from equivalently as MDP reward.

4.3.5 Experiments for further sequential TAMER+RL analysis

I now describe these further sequential TAMER+RL experiments. We first confirm that the newer, Java codebase does not produce qualitatively different results from Section 4.3.1 in mountain car. We then evaluate the algorithms' effectiveness on a different task. Additionally, we analyze the results at a range of combination parameter values (β values) to identify challenges to setting β's value without prior testing.

Using the same $\hat{R}_H$ representation, a linear model of RBF features; task settings; SARSA(λ) parameters; and training records from the TAMER+RL experiments described in Section 4.3.1,[6] we repeat the experiments on the mountain-car task, using all four combination techniques found to be successful in Section 4.3.2 and a range of β combination parameters. We then test these TAMER+RL techniques on a second task, cart pole (shown in Figure 4.6), using an $\hat{R}_H$ model trained by an author. We again use SARSA(λ), choosing parameters[7] that perform well but sacrifice some performance for episode-to-episode stability and the ability to evaluate policies that might otherwise balance the pole for too long to finish a run. The $\hat{R}_H$ for cart pole was learned by k-nearest neighbor (kNN) with a separate kNN model for each action, where k is the square root of the number of samples gathered for the corresponding action. Additionally, the $\hat{R}_H$ for cart pole is learned with a different credit assignment scheme than in previously described TAMER+RL experiments (Section 4.3.1). The previous $\hat{R}_H$'s were learned with the earlier individual reward crediting technique, whereas the $\hat{R}_H$ for cart pole used in this section and all $\hat{R}_H$'s in Section 4.4 were learned with the more current aggregate reward crediting

[6] The models we create—$\hat{H}_1$ and $\hat{H}_2$—from the original training trajectories perform a bit better than did the corresponding models under the original TAMER+RL implementation, which points to one or more small implementation differences. The only difference we observe is that the linear model of $\hat{R}_H$ (and Q as well) contains an additional feature of constant value 1.

[7] Following the notation I used previously for mountain car, the SARSA(λ) parameters for cart pole are $\{\gamma = 0.999, \alpha = 0.05, \lambda = 0.86, \epsilon = 0.085, \sigma^2 = 0.13, j = 8192\}$. These parameters were chosen by decreasing the α value from the best-performing parameter set according to hill-climbing parameter search algorithm similar to that used to obtain parameters for mountain car. Decreasing α increased episode-to-episode performance consistency but slowed the agent's increase in mean performance. For each of the 2 actions, the means of the 4-dimensional RBFs are placed along an $8 \times 8 \times 8 \times 8$ grid in state space. An additional feature of constant value 1 was added. We also hand-tuned certain parameters before automated tuning: SARSA(λ) used replacing traces, ϵ was annealed after each episode by a factor of 0.9995, and the weights of the linear model of Q were each initialized to 0, creating a pessimistic output of 0 for any state-action pair.

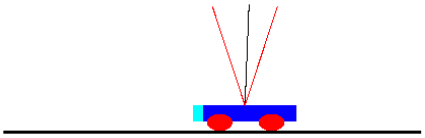


Figure 4.6: A screenshot of the cart-pole task. The dark blue cart is accelerated left and right with the goal of keeping black pole between the red angular boundaries and the cart from moving too far left or right, beyond the edges of the track. The action is shown by the vertical light blue bar.

technique. Both crediting techniques are described fully in Chapter 3. For both tasks, SARSA(λ) uses a linear model with Gaussian RBF features to represent the Q-function and initialize Q pessimistically, as was found effective in Section 4.3.2. In these and later experiments, $\hat{R}_H$ outputs are typically in the range [-2, 2]. The β parameter for reward shaping, action biasing, and control sharing is annealed by a factor of 0.97 at the end of each episode for both tasks.

Like mountain car, the cart-pole task is also adapted from RL-Library [94]. Recall that in mountain car, the goal is to quickly move the car up a hill to the goal. The agent receives -1 reward for all transitions to non-absorbing states. In cart pole, the goal is to move a cart so that an attached, upright pole maintains balance as long as possible. The agent receives +1 reward for all transitions that keep the pole within a specified range of vertical. Cart pole has four state variables: the pole's angle and angular velocity and the cart's position and velocity. There are two actions, accelerating $+c$ or $-c$.

We evaluate each combination technique on four criteria; as in the previously described TAMER+RL experiments, full success requires outperforming the corresponding $\hat{R}_H$'s TAMER-only policy and SARSA(λ)-only both in end-run performance and cumulative reward (or mean reward across full runs, equivalently).

4.3.6 Results and discussion for further sequential TAMER+RL analysis

Figures 4.7 and 4.8 show the results of our experiments for sequential TAMER+RL. For now, I only show results for the β combination parameters that accrue the highest cumulative reward for their corresponding technique.

Qualitatively, our mountain car results reproduce those in Section 4.3.2. Action biasing and control sharing succeed on all four criteria and significantly outperform other techniques in cumulative reward. Reward shaping and Q augmentation also improve over SARSA(λ)-only by both metrics and over the TAMER-only policies in end-run reward.

On cart pole, action biasing and control sharing again succeed fully. This time, Q augmentation also meets the four criteria for success, though it performs significantly worse than action biasing and control sharing. Most interestingly, reward shaping, at its best tested parameter, does not significantly alter SARSA(λ)'s performance on either metric.

By choosing the best β parameter value for each technique, the TAMER+RL experiments thus far reported sidestep the issue of how to use an effective value without first testing a range of values, which would erase the gains in learning speed if parameter testing is counted as part of learning. With experiments in two tasks, we can begin to address this problem by examining each technique's sensitivity to β parameter changes and whether certain ranges of β are effective across different tasks. Figure 4.9 shows the mean performance of each combination technique as β varies. Examining the charts, we consider several criteria:

- performance at worst β value,
- range of beneficial β values,
- and existence of β values that are effective across tasks.

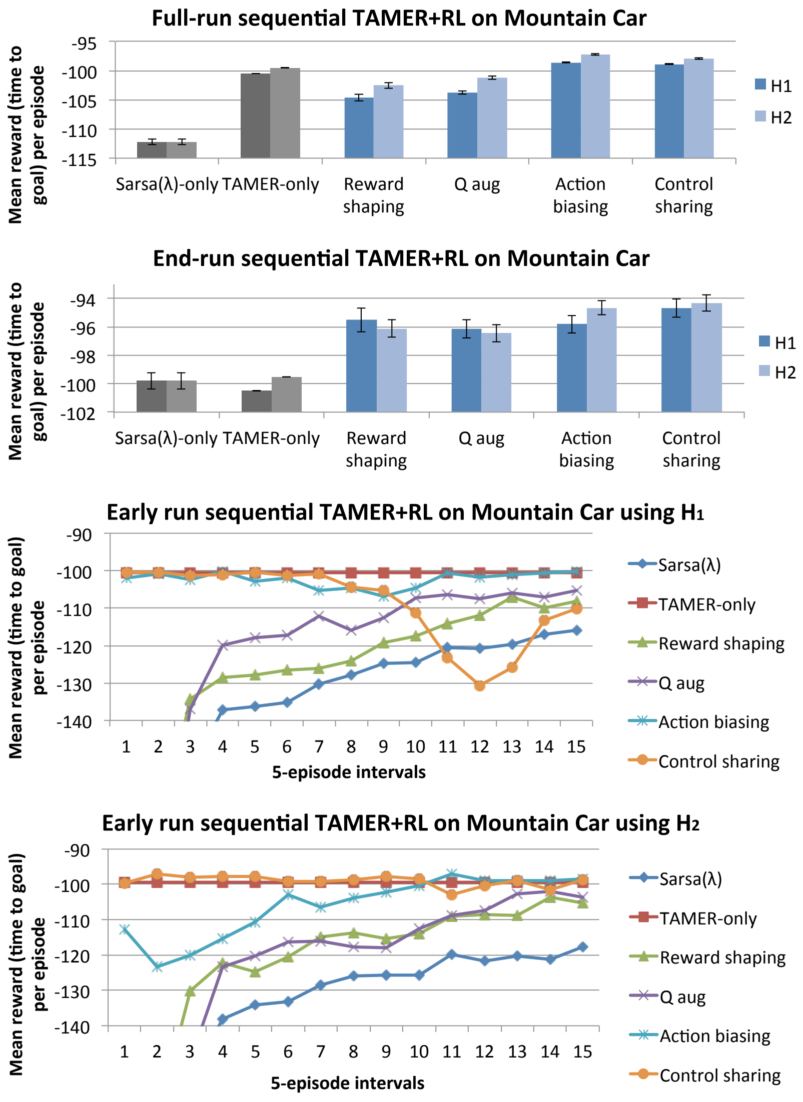


Figure 4.7: A comparison of TAMER+RL techniques with SARSA(λ) and the TAMER-only policy on mountain car over 40 or more runs of 500 episodes. $\hat{H}_1$ and $\hat{H}_2$ are models from two different human trainers. The top chart considers reward over the entire run, and the second chart evaluates reward over the final 10 episodes. Error bars show standard error. The third and fourth charts display mean performance using $\hat{H}_1$ and $\hat{H}_2$ early in the run, during the first 75 episodes.

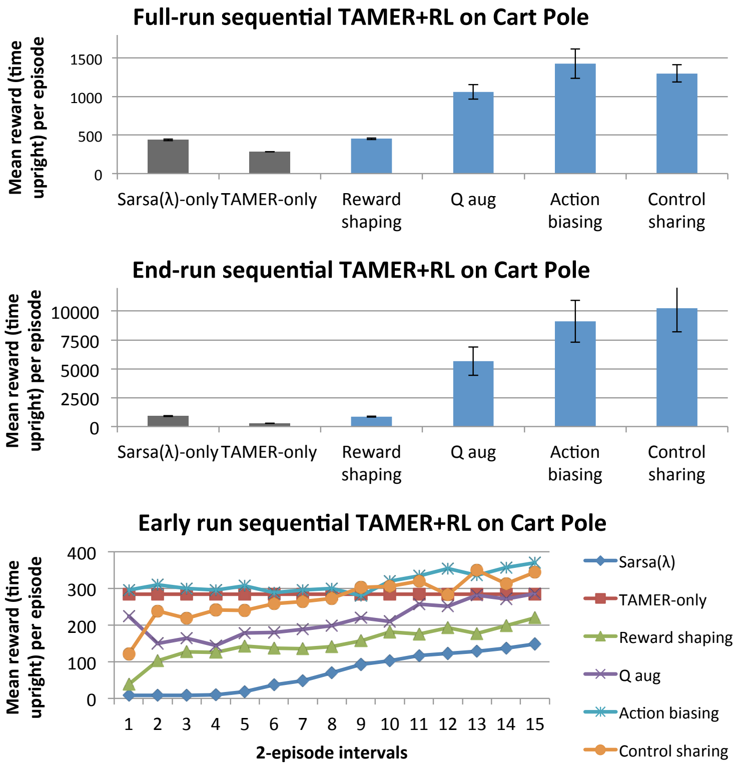


Figure 4.8: The same TAMER+RL comparisons as in Figure 4.7, but on cart pole over runs of 150 episodes. A single $\hat{R}_H$ was used. End-run performance for cart pole is the mean reward during the last 5 episodes.

Evaluating the techniques on these three criteria creates a consistent story that fits with our analysis of the techniques at their best β parameter values (in Figures 4.7 and 4.8). The two combination methods that only affect action selection—action biasing and control sharing—emerge as the most effective techniques without a clear leader between them, and they are followed by Q augmentation and then shaping rewards.

From an RL perspective, the weakness of reward shaping may be counterintuitive. When researchers have combined human reward with RL in the past literature, they have used reward shaping [105, 101], possibly because human re-

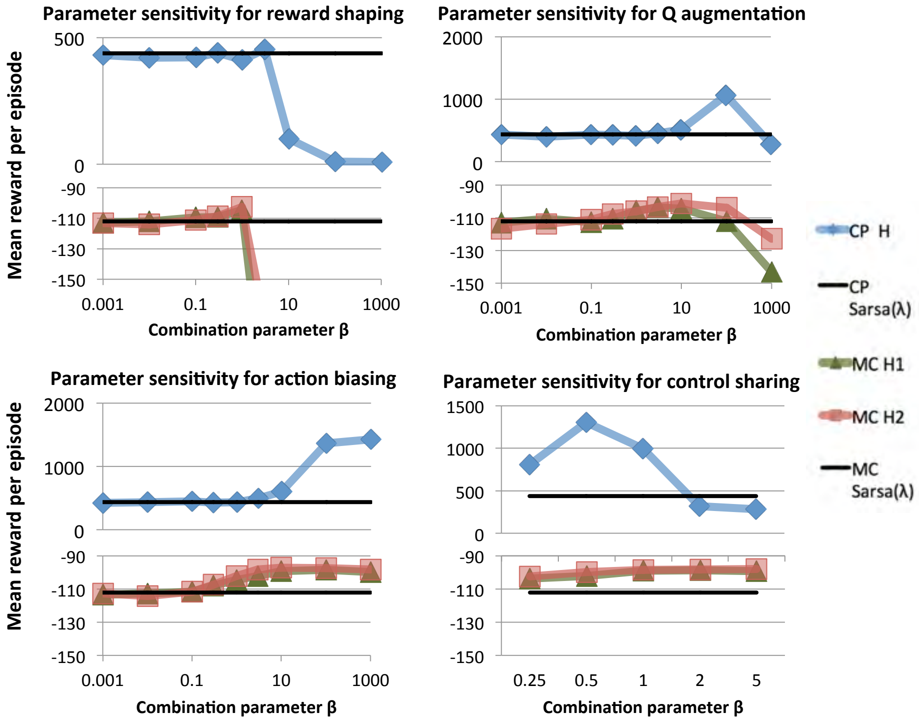


Figure 4.9: Performance of each technique with each tested $\hat{R}_H$ over a range of β parameters on two tasks: cart pole (CP) and mountain car (MC). Note changes in y-axis scaling. ("CP H" denotes learning with the $\hat{R}_H$ for cart pole.)

ward is seen as an analog to MDP reward that should be used similarly. However, though reward shaping is generally cast as a guide for exploration, it only affects exploration indirectly through precariously tampering with the reward signal. Additionally, in Chapter 6 we show that learning action values with $\hat{R}_H$ as the reward function and a discount factor of 1 is highly problematic given the characteristics of human reward. Following the "unified framing of reward shaping and action biasing" in Section 4.3.4, reward shaping is equivalent to adding two Q-functions together, one from MDP reward and one from predictions of human reward, using the same discount factor, which is 1 in these experiments. Therefore, reward shaping creates the same problems that we identify in Chapter 6.

Action biasing and control sharing affect exploration directly, without manipulating reward. Thus, they achieve the stated goal of reward shaping while leaving the agent to learn accurate values from its experience. Following this line of thought, Q augmentation is identical to action biasing during action selection, boosting each action's Q-value by the weighted prediction of human reward. In addition to this direct guidance on exploration, Q augmentation also changes the Q-value during the SARSA(λ) update's calculation of temporal difference error. As discussed in Section 4.3.4, Q augmentation is nearly equivalent to a form of reward shaping called look-ahead advice [114]. Additionally, Q augmentation assumes certain compatibilities between Q and $\hat{R}_H$ that may not exist. In the learning curves in Figures 4.4(a) and 4.8, the agent's performance starts relatively high and quickly decreases and does not surpass the original performance for at least 10 episodes. We suspect that this dip in performance occurs because the agent is moving from using one policy representation, $\hat{R}_H$, to differentiate between actions to another policy representation, Q, and the agent must unlearn the policy from $\hat{R}_H$ to accurately approximate return with Q.

In short, we observe that the more a technique directly affects action selection, the better it does, and the more it affects the update to the Q-function for each transition experience, the worse it does. Q augmentation does both and achieves performance between those techniques that do only one. Further, both techniques assume certain compatibilities between the two reward functions. Q augmentation assumes that $\hat{R}_H$ is informative in a specific way about the differences in return between actions; reward shaping assumes that the predictions of human reward can be used equivalently to the MDP reward.

Taken altogether, these experiments validate our conclusions from Section 4.3.2 and yield new, firmer conclusions about the relative effectiveness of each technique, endorsing action biasing and control sharing over the two other previously successful techniques. And more generally, these results endorse (1) manipulating action selection directly and (2) leaving the action-value model's update unmolested.

These two conclusions endorse a conservative approach in which only the RL algorithm's experience is changed by $\hat{R}_H$. With even further exploration of the characteristics of human reward, as in Chapter 6, we may find other ways to learn the Q_H defined in Section 4.3.4 that are less conservative but more effective.

4.3.7 Summary of sequential TAMER+RL

This section (4.3) introduced, tested, and analyzed results from eight TAMER+RL methods for combining a model of human reward $\hat{R}_H$, learned within the TAMER framework, with the reinforcement learning algorithm SARSA(λ). We obtain positive results for a number of the techniques when pessimistically initializing the Q-function. Specifically the combination methods action biasing and control sharing outperform both of their component algorithms—SARSA(λ)-only and TAMER-only—in both cumulative reward and final performance level on two different tasks. These two most successful techniques both manipulate action selection without directly affecting SARSA(λ)'s update of the Q-function.

4.4 Simultaneous TAMER+RL

To this point, we have assumed that the human training finished prior to any reinforcement learning. Such "sequential" learning is sometimes appropriate; for instance, when a difficult-to-simulate MDP reward function is tied to potentially costly learning trials but the agent can train in simulation with human reward without significant cost. An example of this type of task might be a robot that interacts with workers on a factory line; the MDP reward function would be tied to the factory output, but initial training could occur with only human training and replacing the product that will eventually be created with a prop or some cheaper alternative.

However, this sequential assumption is generally limiting. At any time during an RL agent's learning curve, as long as the agent's policy is still sub-optimal, a human trainer could still improve learning by teaching the agent. For example, after numerous learning trials, RL agents often plateau (i.e., "converge") on a local maximum. At this point, a human could teach the agent to transcend the current local maximum, clearing new paths for learning that would otherwise remain hidden indefinitely.

In this section, we investigate how to modify sequential TAMER+RL algorithms to allow a trainer to step in as desired to alter the course of reinforcement learning while it is in progress. We call this scenario and the algorithms that address it "simultaneous" TAMER+RL. Specifically, the agent should learn simultaneously from two feedback modalities—human reward and MDP reward—as one fully integrated system. As in the sequential TAMER+RL approaches, we examine techniques that use only $\hat{R}_H$ from TAMER in the RL algorithm, otherwise leaving the two algorithms as separate modules.

Since TAMER empirically compares most favorably against RL algorithms in early learning, we expect the greatest gains to come from training near the beginning of learning. However, training at any suboptimal point along the learning curve should benefit the agent, and we hope to do little harm if the agent is already performing optimally and the trainer's feedback cannot help.

In Section 4.2 I listed desirable restrictions on the TAMER+RL combination methods, sequential or simultaneous. Here, we identify some characteristics that are specifically desirable for simultaneous learning:

1. *behavioral consistency*: The agent's behavior should not be erratic, making it difficult to give feedback to specific actions.
2. *responsiveness to the trainer*: The agent should quickly and obviously demonstrate that it is learning from human reward to maintain interactivity. Addi-

tionally, quick responses aid a trainer's own process of learning how to teach effectively.

3. *trainer can give feedback to the MDP-only policy*: If a trainer comes in midway through learning, the trainer should be able to *capture* the good aspects of what has already been learned and criticize the negative aspects. In other words, feedback should not immediately dominate MDP-only learning.
4. *training should affect the agent locally*: $\hat{R}_H$'s influence on the RL algorithm's learning and/or action selection should be larger in more recently trained areas of the state-action space and smaller in areas trained less recently.

The first two characteristics are desirable for any TAMER system, with or without RL. However, simultaneous learning—and its inclusion of RL-based action selection during training—presents new challenges for maintaining behavioral consistency. For instance, control sharing abruptly shifts between two policies, which can create erratic behavior with many different actions in a small time period, increasing the difficulty of giving clear feedback. Also note that the second and third characteristics are in opposition; fully responding to the trainer's reward requires abandoning the policy learned by MDP reward. Our module for determining human influence, described in the following section, strikes a balance by ramping up the influence of $\hat{R}_H$ with increased human reward, keeping the RL policy early on.

4.4.1 Determining the immediate influence of $\hat{R}_H$

Simultaneous learning allows human trainers to insert themselves at any point of the learning process. Consequently, $\hat{R}_H$'s influence should increase in areas of the state-action space with recent human reward—but not in areas that have not been targeted with feedback—and decrease in the absence of human reward, leaving the set of optimal policies unchanged in the limit. Thus, we must do more than annealing a combination parameter β, as is done in sequential learning.

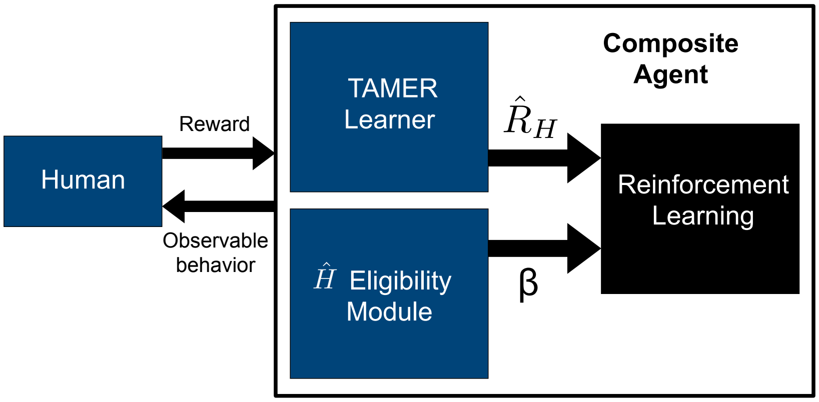


Figure 4.10: A conceptual diagram of simultaneous TAMER+RL. Contrasting with the sequential scenario (Figure 4.1), the human interacts with the agent that results from the combination of $\hat{R}_H$, which can be modified any time, and the RL agent. The task environment and MDP reward are not explicitly shown; the composite agent interacts with the task environment and receives MDP reward.

We determine $\hat{R}_H$'s influence through a novel adaptation of the eligibility traces often used in reinforcement learning [90]. I will refer to our adaptation as the *eligibility module*. Its interaction with TAMER and the RL algorithm are shown in Figure 4.10. The general idea of this eligibility module is that we maintain an eligibility trace for each state-action feature[8] that represents the recency of training while that feature was active (i.e., non-zero). Then, the eligibility traces and a time step's feature vector together calculate a measure of the recency of training in similar feature vectors, as shown in Figure 4.11. That measure is multiplied by a constant scaling parameter c_s to create the β term introduced in Section 4.2. The implementation follows.

Let $\overrightarrow{e}$ be the vector of traces and $\overrightarrow{f}$ be the feature vector, where feature extraction is defined such that feature values exists within the range $[0, 1]$, feature values correlate with an intuitive level of feature activation, and the feature vector

[8] The feature vector is extracted from the current state-action pair; each feature value should range between 0 and 1. We advise using features that generalize across state space (e.g., Gaussian RBFs). The state-action features need not match those of either $\hat{R}_H$ or Q.

is always non-zero. In our experiments described in the next section, we fit these constraints with scaled features from Gaussian RBFs and one constant feature. The eligibility module can be viewed as a function over $\overrightarrow{e}$, $\overrightarrow{f}$, and c_s that outputs β within the range $[0, c_s]$. A guiding design constraint is that when $\overrightarrow{e} = \overrightarrow{1}$ (i.e., each trace in $\overrightarrow{e}$ is the maximum allowed), the normalized dot product of $\overrightarrow{e}$ and any $\overrightarrow{f}$, denoted $n(\overrightarrow{e} \cdot \overrightarrow{f})$, should equal 1 (since it weights the influence of $\hat{R}_H$). To achieve this, we make $n(\overrightarrow{e} \cdot \overrightarrow{f}) \; = \; \overrightarrow{e} \cdot (\overrightarrow{f} \; / \parallel \overrightarrow{f} \parallel_1) \; = \; (\overrightarrow{e} \cdot \overrightarrow{f}) \; / \; (\parallel \overrightarrow{f} \parallel_1) \; = \; \beta \, / \, c_s$. Thus, at any time step with normalized features $\overrightarrow{f}$, the influence of $\hat{R}_H$ is calculated as $\beta = c_s(\overrightarrow{e} \cdot \overrightarrow{f})/(\parallel \overrightarrow{f} \parallel_1)$. This formula has a desirable mathematical characteristic; for a given $\overrightarrow{e}$, β is higher when relatively large feature values correspond to large trace values—indicating the current state-action pair is similar to the recently trained state-action pairs—and β is smaller when large feature values correspond to small trace values.

We decay the traces at each time step, $\overrightarrow{e} := \mathit{stepDecayFactor} * \overrightarrow{e}$. Using a different decay factor, the same operation occurs at the end of each episode. If training is occurring, the traces are updated after they are decayed using accumulating traces capped at 1: $e_i := min(1, e_i + (f_i * a))$, where e_i and f_i are the i^{th} elements of $\overrightarrow{e}$ and $\overrightarrow{f}$, respectively, and a is a constant factor that moderates the speed of accumulation.

Though this eligibility module is inspired by eligibility traces used in TD(λ), it differs from eligibility traces in several key ways. This module maintains a vector of traces similarly to how TD(λ) maintains eligibility traces. However, unlike eligibility traces, it only increases the traces during training. In addition, rather using the traces to determine the extent to which each feature's corresponding Q-value parameter is updated, we use them to output a measure in $[0, 1]$ that roughly indicates how recently nearby states have been trained.

A video demonstration of the eligibility module can be viewed online at the

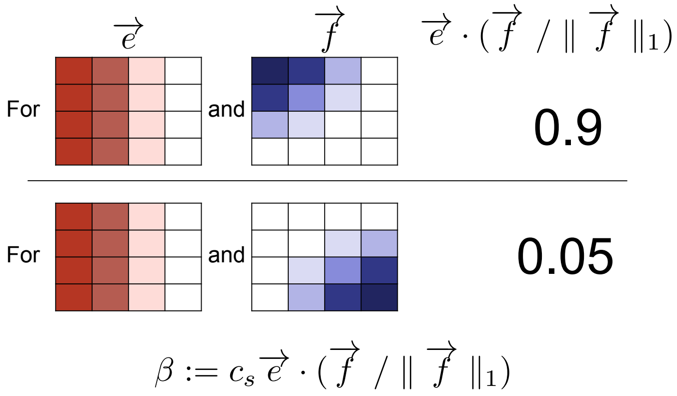

$\overrightarrow{e}$
$\overrightarrow{f}$
$\overrightarrow{e} \cdot (\overrightarrow{f} \ / \parallel \overrightarrow{f} \parallel_1)$
For
and
0.9
For
and
0.05
$\beta := c_s \overrightarrow{e} \cdot (\overrightarrow{f} \ / \parallel \overrightarrow{f} \parallel_1)$

Figure 4.11: A simple graphic illustration of the calculation of $\overrightarrow{e} \cdot (\overrightarrow{f} \ / \parallel \overrightarrow{f} \parallel_1)$ = $(\overrightarrow{e} \cdot \overrightarrow{f}) \ / \ (\parallel \overrightarrow{f} \parallel_1)$, which is near 0 when the currently "active" features have not been active during recent training and is near 1 when these features have been. Here, consider $\overrightarrow{f}$ to be a set of Gaussian radial basis functions that form a 4 × 4 grid over a 2-dimensional state space. (For simplicity, the action is not considered here.) In the top scenario, the state is somewhere in the top left square and the active features overlap heavily with recently trained state. Thus, the output is near one, possibly 0.9. In the bottom scenario, the state is in the bottom-right square where there is less overlap, resulting in a lower output such as 0.05.

following URL: `http://www.cs.utexas.edu/~bradknox/papers/tamerrl`.

4.4.2 Simultaneous learning experiments

Our experiments test the effectiveness of simultaneous TAMER+RL when training starts either at the beginning of learning or after significant learning. We again use the mountain car and cart pole tasks, and we focus on the two best-performing combination techniques, action biasing and control sharing. For both tasks, SARSA(λ) uses the same Q-function representation and parameters as in Section 4.3.5. Also, $\hat{R}_H$ is learned by k-Nearest Neighbors for both tasks, as it was for cart pole in Section 4.3.5.

In the eligibility module, features are extracted almost identically as they are for SARSA(λ). There are three differences. First, the outputs of the Gaussian RBFs are scaled to the range $[0,1]$. Second, β's application in control sharing cannot be action-specific, so the eligibility module's features for control sharing are a single grid over the state space (not one grid per action). Third, the RBF width $\sigma^2 = 0.08$ for cart pole.

The scaling parameter c_s used to determine β for mountain car and cart pole is respectively 100 and 200 for action biasing and 2 and 1 for control sharing. These values were chosen to be near the upper end of each combination method's effective β values in Figure 4.9. The accumulation factor a for eligibility is 0.2. For mountain car, the eligibility module's traces are annealed by a factor 0.9998 each time step and are not annealed at the end of each episode; for cart pole, traces were annealed by 0.99996 each step and 0.98 after each episode.

Training in mountain car occurs either for 16 episodes, starting at episode 1, or for 12 episodes after 20 episodes of SARSA(λ)-only learning. In cart pole, training at start occurs for 12 episodes, and training after 25 episodes of SARSA(λ)-only learning lasts 8 episodes. The start times are chosen to represent the beginning of learning and also a point at which the SARSA(λ) agent has learned a policy that is much improved but still quite flawed.[9] The number of episodes corresponds to an informal assessment of how many episodes are needed to satisfactorily train the agent; training at later start times progresses more quickly. The trainer (the author) has a button that starts and stops training during the designated training episodes, letting the human observe without the agent updating $\hat{R}_H$ or increasing any traces within the eligibility module. At all times, whether training is occurring or not, the agent continues to learn a Q-function.

An added experimental challenge is that the training is inextricably bound to one specific run, whereas sequential experiments can reuse the same training ses-

[9] Note that sequential TAMER+RL differs from the case of simultaneous TAMER+RL where training occurs at the start because the sequential algorithm begins with a pre-trained $\hat{R}_H$. The training episodes are not counted in sequential TAMER+RL experiments.

sion for any number of parameters and combination techniques. Thus the depth of analysis that can be done for a set number of trainer-hours is much more limited with simultaneous TAMER+RL. Mountain car and cart pole training sessions typically took around 8 minutes and 15 minutes each, respectively. Consequently, each experimental condition was limited to 3 runs of training for a total of 12 runs on each task.

4.4.3 Simultaneous learning results and discussion

The results of our simultaneous TAMER+RL experiments are shown in in Figures 4.12 and 4.13. Though the sample size is too small to show statistical significance, there is a clear pattern of both action biasing and control sharing outperforming SARSA(λ). The condition that is closest to SARSA(λ) in terms of standard error, control sharing on cart pole where training begins after 25 episodes, still receives almost twice the reward of SARSA(λ). We also observe that training at the beginning of learning is more effective—in terms of mean reward through a run—than training after some MDP-only learning, as we expected.

Seeing that training is most effective at the start of learning, one might ask whether the n episodes of MDP-only learning before training is helping or whether the prior learning should be abandoned to start from scratch. We can quantitatively evaluate this question. Resetting after n episodes is the same as simply training from the start and stopping n episodes early. So if we ignore the first n episodes of the later-training group and the last n episodes of the training-at-start group, the comparison of the groups' mean reward addresses this question. In other words, for a task with run size m, we examine two conditions per combination technique: (1) from the trajectories where training began at the first episode, the performance of the the agent from episodes $(1, 2, ..., m-n)$ is averaged, and (2) from the trajectories where training began at episode n, the performance of the agent during episodes

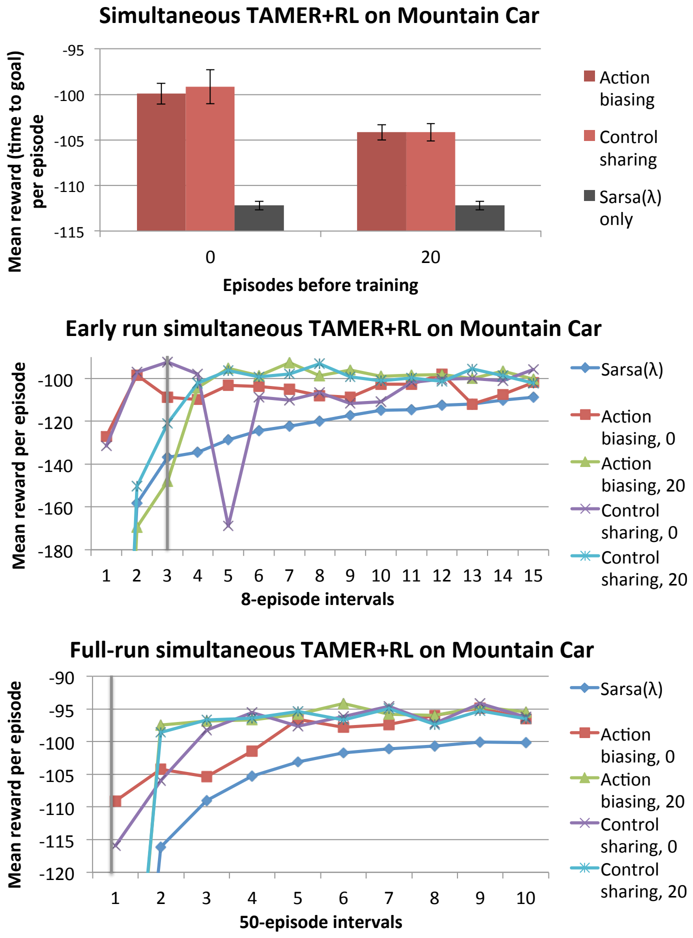


Figure 4.12: Simultaneous TAMER+RL results on mountain car. Unlike sequential TAMER+RL, performance during training episodes is counted. In the top graph, mean reward is calculated over runs of 500 episodes in mountain car and 150 episodes in cart pole. Standard error is shown. In the lower two plots, learning curves are shown at two different scales: the top plot shows mean performance in the earlier episodes of the run and the bottom plot shows mean performance over the entire run. The number in each legend entry indicates the episode number at which training started. A vertical gray bar is placed at the point where the later training period started, the 20th episode.

$(n+1, n+2, ..., m)$ is averaged. Thus, each condition is examined over $m-n$ episodes, and training begins at the first episode of examination. The main difference between

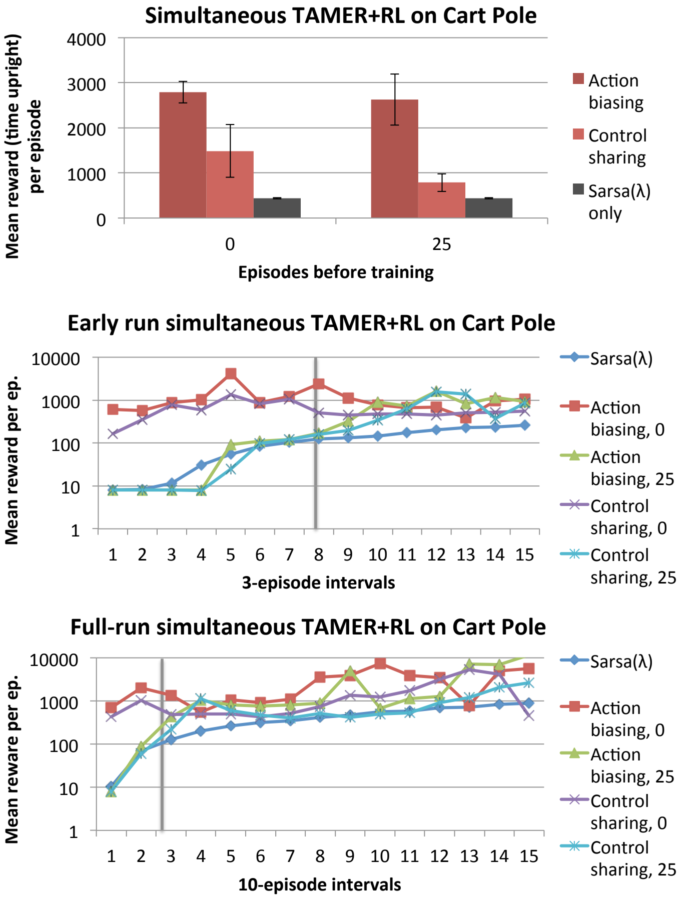


Figure 4.13: Learning curves for simultaneous TAMER+RL on cart pole, following the same format as Figure 4.12.

the conditions is that the later-training group (i.e., starting at n) starts training after already learning from MDP reward, so we can reasonably conclude that performance differences arise from the presence or lack of MDP-only learning prior to training.

Of four such comparisons (2 techniques x 2 tasks, shown in Figure 4.14), the later-training group outperforms three times and is roughly equal once, suggesting that the prior learning does indeed help.[10]

[10] The only other known difference between conditions is that agents with prior learning were given less episodes of training than those that were trained, as was described earlier in this section. Despite the apparent disadvantage of fewer training episodes, these agents still outperform those

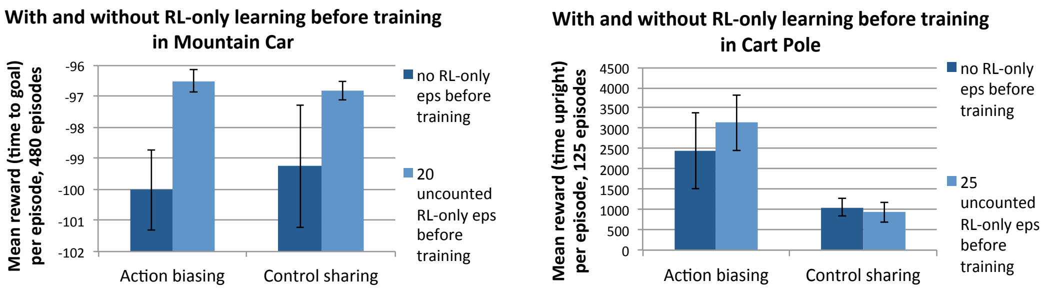


Figure 4.14: A comparison of simultaneous TAMER+RL performance with and without MDP-only learning before training. The lighter blue bars indicate the mean performance of agents that received 20 or 25 episodes of MDP-only learning before the human began teaching. The episodes of prior learning are not counted towards mean reward. The agents corresponding to darker blue bars had no such learning prior to training. Standard error is shown.

For clarity, I note that we do not aim to quantitatively compare sequential and simultaneous TAMER+RL. Our results in Figure 4.14 conclusively show the benefit of training after some MDP-only learning, when it's too late to learn sequentially. Therefore, simultaneous learning provides benefits that sequential learning cannot. And when training without MDP reward is relatively costless, sequential learning allows an agent to be thoroughly taught before beginning more costly learning with MDP reward; thus, sequential learning likewise provides benefits that simultaneous learning cannot. Neither learning scenario is strictly better than the other.

These results, shown in Figures 4.12, 4.13, and 4.14, demonstrate the potential effectiveness of simultaneous TAMER+RL with the eligibility module.

4.5 Related work on transfer learning for RL

TAMER+RL is a novel learning paradigm; this research is the first to specifically consider how to combine a model of human reward with MDP reward. As such, there is little work that solves exactly the same problem. However, from a higher-

without prior learning in the subset of episodes examined here.

level perspective, it fits within the general research area of transferring knowledge to aid reinforcement learning. In this section, I review past work on transferring knowledge to aid reinforcement learning, relating various works to methods that we tested for TAMER+RL in this chapter. I first look at transfer from general sources of knowledge and then from humans. For related work on agents learning from human teachers without MDP reward, I refer the reader to Chapter 3.

4.5.1 General transfer

Transfer learning for reinforcement learning typically focuses on how to use information learned in a source task to improve learning in a different target task. Our type of transfer differs: the task stays constant and we transfer from one type of task knowledge (an $\hat{R}_H$ function) to a different type (a Q function).

Some approaches create inter-task mappings from a source MDP to a target one. Such mappings have been used in the target task to initialize the Q function [99], as we did in Method 3, and to use recorded (s, a, r, s') experience tuples from the source task to estimate the transition and reward functions for model-based RL [97]. Other approaches derive a policy from the source task and use that policy to guide learning in the target task. For example, Fernandez and Veloso [29] created an agent that uses a policy from a previous task to aid learning in a task that differed only in its reward function. The agent learns whether to choose from the old policy, choose from the policy it was currently learning, or explore. This approach is also similar to our Method 5. From a policy π_s derived from a learned source task, Taylor and Stone [98] compare the results of three techniques: adding $\pi_s(s)$ as an extra action (similar to Method 5), increasing the Q-value of the $Q(s, \pi_s(s))$ (similar to Method 4), and adding a state variable that takes the value $\pi_s(s)$ (similar to Method 2).

Other forms of transfer can occur within the same task. *Reward shaping* [24, 65] is changing the output of the reward function to learn the same task, as we did for the technique of the same name (Method 1 in Section 4.2). The difference of the shaped reward function and the original one can be seen as the output of a shaping function. With a few assumptions, Ng et al. [70] prove that such a function f, if static, must be defined as $f = \phi(s') - \phi(s)$, where $\phi : S \rightarrow \mathbb{R}$, to guarantee that shaping the reward function will not change the set of optimal policies. Method 8 is a good-faith attempt to convert $\hat{R}_H$ to a potential function for reward shaping. Additionally, the reward shaping TAMER+RL method anneals the output of its shaping function, avoiding Ng et al.'s theoretical constraint.[11] Whitehead and Ballard

This section and Section 4.3.4 situate each of the eight combination methods within related work. Though all methods at least share similarities with previous work, some are unique and novel, as is transferring from a model of human reward to a reinforcement learning algorithm.

4.5.2 Transfer from non-expert humans

Sridharan [87] extended some of the work reported in this chapter (and reported previously in the work that this chapter is built upon [48]), showing that TAMER+RL action biasing improves an RL agent's learning in the domain of 3v2 keepaway. He tests a "bootstrapping" approach for determining β that sets it by a metric of agreement between the Q-function policy and the $\hat{R}_H$-policy, where more agreement yields a larger β.

The only other algorithms for agents to learn from both human and MDP reward employ a form of reward shaping (which we introduce in Section 4.3.4) [107, 101]. Each of these projects is described in Section 2.5.1. A key difference in their approaches and the reward shaping technique in this chapter is that their algorithms

[11] However, mountain car violates one of their assumptions, as there are mountain car policies that never reach the absorbing goal state.

directly apply the human reward value to the current reward (instead of modeling reward and using the output of the model as supplemental MDP reward).

Judah et al. consider a learning scenario that alternates between “practice”, where actual world experience is gathered, and an offline labeling of actions as good or bad by a human critic [38]. Using a probabilistic technique with a few assumptions, the human criticism is input to a loss function that lessens the expected value of candidate policies while also automatically determining the level of influence given to the criticism. From some mixed results and comments from frustrated subjects, they predicted that redesigning their system to be more interactive and to let the human train periodically—characteristics of simultaneous TAMER+RL—would improve performance.

Other recent work has employed non-expert humans to aid RL algorithms. Subramanian et al. [89] asked humans to identify useful options (a certain type of high-level action composed of multiple base-level actions) and features relevant to those options. Subjects then gave demonstrations of each option behavior, which were used to learn policies for the options. Cobo et al. [21] choose state features from their ability to predict actions in human demonstration data. With these features, the performance of a monte-carlo RL algorithm is greatly improved over simply regressing to imitate human actions given state and over the RL algorithm on raw state. In more recent work, Cobo et al. [20] additionally use human demonstrations to divide the state space of an MDP into subtasks and create a distinct state abstraction for each subtask. Kuhlmann et al. [57] created a system that incorporates natural-language advice (e.g., ”When in X situation, do Y.”) into a reinforcement learning agent by providing a state-action feature for each piece of advice that indicates whether that state-action pair would violate the advice, would follow the advice, or is irrelevant to the advice.

Learning from demonstration has also been used to improve reinforcement learning, using preprogrammed policies [113, 78] or humans [95, 86] to provide demonstrations for an agent that observes and learns. These approaches are similar to control sharing (Method 7 in Section 4.2). An advantage, though, of human reward over demonstration is that human reward permits learning the relative values of actions, allowing techniques like action biasing (Method 6 in Section 4.2) to gently push the behavior of the RL agent towards the policy endorsed by $\hat{R}_H$, whereas pure demonstration is all or nothing—either the demonstrator or the learning agent chooses the action. Additionally, trainers can reward state-action pairs visited by the agent's policy, whereas demonstrations might not ever visit areas of the state space that the learning algorithm visits.

4.6 Summary

TAMER+RL learns from human reward—a direct but flawed evaluation of behavior—and predefined MDP reward—which specifies optimal behavior through indirect evaluation. Combined, these two complementary sources of reward can produce performance superior to what is learned from either source alone.

In this chapter, we test eight combination techniques in a sequential learning setting, where RL follows human training, and two techniques emerge to consistently yield the greatest benefit to the RL algorithm: action biasing and control sharing. Both techniques affect the RL algorithm only during action selection, adding a bias towards experiencing human-approved state-action pairs. We further adapt these two techniques to a simultaneous learning scenario using an eligibility module to determine the influence of the human reward function over time and state-action space. The simultaneous versions also successfully improve learning over the RL algorithm alone, whether training starts at the beginning of learning or after considerable MDP-only learning.

We examine combining TAMER with value-function based algorithms in this chapter. Some of the combination techniques presented here would likely improve policy-search algorithms, such as Method 2 or an adaptation of Method 3 that uses $\hat{R}_H$ as the initial policy function. Intriguingly, Sridharan found that TAMER+RL with action biasing improved the performance of a policy-search algorithm, cross entropy, on Tetris [87].

This chapter ends this thesis' investigation of TAMER+RL methods. In the following core chapters (5 and 6), we again consider the pure interactive shaping problem without MDP reward. However, Chapter 6 sheds light on the relative weakness of reward shaping, which I discuss as one of the future work items in Section 7.1.

Chapter 5

How Humans Teach Agents: A New Experimental Perspective

To effectively design agents that leverage available human expertise, we need to understand how people naturally teach. In this chapter, I describe two experiments that ask how differing conditions affect a human teacher's feedback frequency and the computational agent's learned performance.[1] *For both experiments,* TAMER *agents fulfill the pupil's role. The first experiment considers the impact of a self-perceived teaching role in contrast to believing one is merely critiquing a recording. The second considers whether a human trainer will give more frequent feedback if the agent acts less greedily (i.e., choosing actions believed to be worse) when the trainer's recent feedback frequency decreases. From the results of these experiments, we draw three main conclusions that inform the design of agents. More broadly, these two studies continue a growing tradition of using agents as highly specifiable social entities in experiments on human behavior, extending this paradigm to learning agents.*

[1] The this chapter is adapted from published work by the author and his advisor in collaboration with Brian D. Glass, Bradley C. Love, and W. Todd Maddox [46].

This thesis aims to create agents with relatively natural interfaces that allow anyone—including non-programmers—to guide the learning process of an agent. But to effectively design such agents, we need to understand how people naturally teach. In this chapter, we ask how human teachers can be affected by changes in their beliefs and in the pupils' behaviors. This chapter is distinguished from the other chapters of this thesis for putting the human trainer in focus, not the agent's algorithm.

Specifically, we examine effects on the frequency with which teachers give feedback and the quality of the behavior learned from their feedback. I describe two experiments, each of which addresses this question by using a computational agent as the pupil of a human subject. The agents are built within the TAMER framework, as presented in Chapter 3 and summarized for readers' convenience in Appendix A. For these experiments, we vary the conditions under which the humans teach and then look for differences in training statistics and agent performance.

In what we call the *critique experiment*—a test of the impact of taking on the role of teacher—there are two conditions: one in which subjects know the agent is learning from their feedback and another in which subjects believe they are merely critiquing a recording of a learning agent. We predicted that the participants' assumed roles would affect what the agents learn from the resulting feedback and the frequency at which trainers give feedback. Against our intuitions, the results of the critique experiment indicate that changing the trainer's role has little effect on these two dependent variables. These results suggest that either the manipulation was ineffective, the quality of the trainers' feedback was not greatly altered by whether they considered themselves to be teaching, or the learning agents were unaffected by such changes in training.

Attempting to directly study the relationship between trainer engagement and the agent's learned task performance, we conducted a second experiment wherein the agent in one condition directly responds to changes in recent feedback frequency.

This experiment, called the *feedback-frequency experiment*, considers whether a human trainer will give feedback more frequently if the agent acts less greedily (i.e., "exploring" or choosing actions believed to be worse) when the trainer's recent feedback frequency decreases. The results indicate that tying non-greedy action to a trainer's feedback frequency increases the overall frequency—and thus, the number of learning samples available. However, the effect on performance is unclear, which I discuss later in Section 5.3.2. The feedback-frequency experiment yields two contributions that inform the design of agents than can learn from human teachers.

First, these results provide a strategy for increasing trainer engagement—lowering performance, especially when engagement drops—that could be incorporated in any agent that learns from a human teacher. Traditionally, learning agents receive feedback from encoded objective functions, called "reward functions" in reinforcement learning (RL) [90]; reward functions give regular feedback after each discrete time step. But human teachers are more complex than an encoded objective function—creating new challenges for learning—and yet can be more effective, especially given their ability to adapt to their pupils. The experiment described here adds to the currently small base of knowledge on how to create agents whose learning algorithms and behavior are designed with a respect for human trainers' strengths and limitations.

The second contribution of the feedback-frequency experiment is a proof-by-example that the common practice of categorizing all actions as either exploitation—greedily choosing actions currently thought to be best for the task—or exploration—trying other actions to learn whether they are actually superior—is insufficient when a human is in the learning loop. Since the human is reacting and adapting to the agent, the agent can take actions to intentionally affect the human's behavior. Rather than exploiting to get the highest appraisal or exploring to try new actions, the agent's actions might instead be used to communicate to, or even reinforce behavior of, the human trainer.

Additionally, this chapter comprises a more general contribution. Social agents, including social robots, provide an emerging opportunity to study human social behavior [69, 14, 22]. A computational agent's behavior can be parametrized and recorded much more thoroughly than can a human's behavior. Thus such studies allow more controlled conditions at the potential cost of less authentic interactions, yielding a different perspective from studies that use humans opposite the subjects, a perspective that has its own strengths and weaknesses. As our final contribution of this chapter, these experiments illustrate this new experimental method, providing an instantiation of the previously unexplored version in which the agent learns during an interaction that is itself affected by the agent's learning (i.e., socially-guided machine learning [104]). Also, I discuss the motivation for studying human behavior through human-agent interaction in the context of these experiments.

The remainder of the chapter is organized as follows. In Section 5.1, I discuss related work and then describe the learning paradigm and algorithm used in the experiment. Section 5.2 explains the experimental designs and results, which are then discussed in Section 5.3 along with our observations from the general practice of studying human behavior with interactive agents.

5.1 Related work and background

In this section, I first motivate our experiments by discussing related work in Section 5.1.1. Then in Section 5.1.2 I provide brief background on TAMER and the task that the TAMER agents are taught.

5.1.1 Related work

Agents learning from human teachers

The field of agents that learn from humans is young but already has a rich and varied literature, which I review in Section 2.5. The concept of an agent using actions to affect a human teacher, though usually left out of the conversation about such human-oriented learning agents, has been explored previously. Nicolescu and Mataric [71] speak of "communication by acting," which is using behavior to communicate intentions and needs. They specifically consider how a robot can ask for help after failure and conduct experiments in which the robot repeatedly tries to execute a failed behavior to signal a need for help. A difference between their approach and ours is that, in their work, the human's requested assistance comes after learning, so the robot improves its current performance through assistance but it does not improve its autonomous performance (i.e., when no human is present). In various work by Thomaz and Breazeal, agent *transparency* is emphasized as a way to improve the trainer's mental model of the agent and, consequentially, the quality of training as well. In one study [107], a software agent pauses to gaze at objects affected by potential immediate actions before choosing its action. The number of gazes is determined by the number of actions that the agent has learned to value within some bound of the maximally valued action; thus more gazing signals a higher level of uncertainty. Compared to agents that do not engage in this gaze behavior, the gazing agents received more action guidance from human trainers at times of high uncertainty and less guidance at times of low uncertainty.[2] In other work [106], Thomaz and Breazeal discuss the use of emotive facial expressions, gaze, and gestures to communicate the agent's state to the teacher, providing information to guide subsequent teaching. For example, the agent could communicate the

[2] The gaze behavior apparently creates more time before action—more time to give guidance—which itself may have some confounding effects that could be examined in future research.

unexpectedness of an outcome, acknowledgement of achieving a goal, or the object of the agent's attention.

How humans teach

The general question of how humans teach has been studied extensively. I review some of the more relevant work here.

Some work has specifically examined how humans teach social robots or other agents. Thomaz and Cakmak [108] examined how people teach a robot affordances (i.e., action-effect relationships) to manipulate objects. Among their findings, they observed that humans point out affordances that are rare in systematic exploration of the object configuration and action spaces. They also found that people often remove the object before the robotic action completes, possibly indicating that the remaining part of the action would not have caused a desirable effect. Kim et al. observed how people talk while teaching a robot and found that "people vary their vocal input depending on the learner's performance history" [44]. In work by Koachar et al. [42], human subjects teach a complex task to a fake agent in a Wizard-of-Oz experiment. The teachers could give reward-based feedback, give demonstrations, teach concepts by example, and test the agent's skills. The authors found that "teaching by [feedback] was never employed by itself and in the 82% of cases where it was used, it followed another teaching type in all but 2 cases. 58% of the teachers who used feedback used it exclusively after testing." A consistent finding across all of these studies is that human teachers break implicit and explicit expectations built into the learning system (e.g., removing an object before an action is complete), suggesting that agents should be robust to at least some such violations.

Looking particularly at teaching by explicit reward and punishment, there has been much research on how humans and other animals learn [12] and, complementarily, how people *should* teach [80, 79]. However, little has been said about how people actually *do* teach by explicit reward and punishment and, complementarily, how pupils should learn from it—as this chapter does. One exception is by Thomaz and Breazeal [105], who had people teach a task to a software agent by reward and punishment. They found that people gave more reward than punishment and that people appeared to be using the feedback mechanism to give guidance to the agent, again interestingly breaking protocol.

Studying human social behavior with human-agent interaction

Here I discuss the practice of studying *human-human interaction* using *human-agent interaction* experiments. I do not include in this category studies that draw conclusions that are only of interest to the human-robot interaction or human-agent interaction communities. I save our discussion of the motivation for using agents in lieu of humans for Section 5.3.4, where I can interweave our experimental results.

Replacing humans with agents in experiments on human social behavior has been proposed by numerous researchers [69, 14, 62, 22]. Of the relevant social robotics studies which we are aware, all used both human-human and human-robot interaction [34, 83]. In one [63], people converse with either a human, a Wizard-of-Oz robot (i.e., a robot controlled by a human but pretending to be autonomous), or an openly remote-controlled robot. Researchers examined which direction subjects moved their eyes when breaking eye contact after having been asked a question. The results on the effect of which conversational partner was used were inconclusive, which the authors attribute to high variance and a small sample size. In another study [25], each subject watched two videos, one of a collaborative human assistant and another of a collaborative robot assistant. Afterwards, subjects rated the collaboration on multiple criteria, such as comfort with and trust in the assistant. Subjects were divided along two additional variables. Along one of these variables, subject nationality, results on collaboration ratings were consistent across human and robot versions (e.g., Chinese subjects gave higher trust ratings for both human and robot assistants than did subjects from the U.S.). The ratings were not consistent along the other variable, how strongly the subject is prompted to consider the assistant to be part of her ingroup (i.e., a group that the subject strongly identifies with).

From these studies, we see two patterns. First, the robots and humans were not perfectly interchangeable as social partners. However, the difference in their effects was usually by whether results were significant, not by significant results in opposite directions. And the results did agree a fair amount. Overall, their specific robotic partners created interactions that resembled those with humans in some situations, but not fully. Note, though, that results from studies with human actors following scripted interactions—as the human partners in the above social robotics experiments do—differ in their own way from the ground truth of authentic human-human interaction. The second pattern is that none of these experiments use agents to fully replace humans where their use would be problematic or to perform analysis that would be impossible with humans. Among previous work that employed computational agents or robots to study human interaction, our experiments stand out for random assignment and controls, for the relatively large sample sizes, and for the complexity of our agents.

5.1.2 Background

The TAMER learning agent

The experiments carried out in this chapter involve human subjects training a computational learning agent that implements the TAMER framework, employing the algorithm described in Section 3.2. TAMER is explained fully in Chapter 3 and summarized for readers' convenience in Appendix A. Our experiments in Chapter 3 indicate that humans can often train TAMER agents to perform tasks well (but imperfectly) within shorter time than a traditional RL agent would learn, reducing the costs of poor performance during learning.

The experimental task: Tetris

Subjects observed their agents play Tetris (Figure 5.1)—a well known, computer-based puzzle game—on a computer screen and delivered real-time feedback. The domain and training interface are that of the second Tetris experiment of Chapter 3 and are described in detail in Section 3.4.1. The TAMER Tetris algorithm is described in Section 3.4.1 under the heading "Tetris Instantiation", likewise corresponding to the second experiment discussed there.

5.2 Experimental design and results

In this section, I discuss the designs and results of the two experiments. I first describe aspects of experimental design that were common to both experiments.

We evaluated participants' teaching with descriptive analyses as well as simulations of their learned models' ($\hat{R}_H$s') performances. For descriptive analyses, we considered the human responses' frequency. All descriptive analyses were conducted over time in bins defined by intervals of 80 time steps, where each time step includes the appearance and subsequent placement of a single piece. In other words, the first bin considered time steps 1 to 80, the second considered steps 81 to 160, and so on.

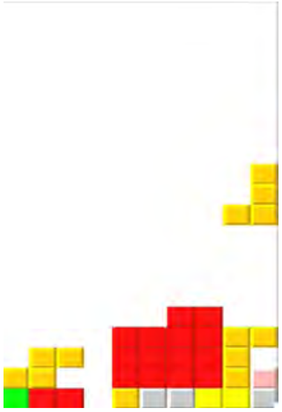


Figure 5.1: A screenshot of RL-Library Tetris.

Simulations were performed offline for each subject at 80 time-step intervals, fixing $\hat{R}_H$ and using a greedy policy—and thus fixing the learned behavior—after 80, 160, ... time steps of training and then testing the fixed behavior's performance over

20 games (i.e., episodes). For our performance metrics, we use the mean number of lines cleared per episode by a TAMER agent over the 20 games in simulation at each time interval. This analysis evaluates the quality of a subject's training by simulating the performance of the fixed policies shaped from their feedback.

Subjects were drawn from the undergraduate community at the University of Texas at Austin through the Department of Psychology.

5.2.1 The critique experiment: teaching vs. critiquing

In our first of two experiments, the *critique experiment*, we tested how donning the role of teacher affected subjects' feedback frequency and the effectiveness of their teaching.

Design

Subjects were randomly assigned to one of two conditions: Teaching or Critiquing.

1. Teaching (n = 27): Subjects are aware that the agent learns from his or her feedback.

2. Critiquing (n = 30): Subjects are told that they should critique a recording of an agent learning.

Our hypotheses about the conditions' effects on feedback frequency and agent performance varied. The dominant hypothesis was that when teaching, humans would satisfice aggressively, dramatically reducing their feedback once the agent appeared to be doing reasonably well. This reduction in feedback might harm the agent's performance compared to one that received a consistent level of feedback. If the non-teaching subjects trained better agents, it would suggest that human trainers need to be fooled into providing large amounts of feedback over time to maximize a TAMER agent's performance. Another intuition was that the Teaching subjects would be more engaged and attentive, leading to a contrasting hypothesis that the teaching group would give more feedback and achieve better performance. The plausibility of either result motivates this experiment.

Results

Our results focus on the question of whether frequency of feedback and agent task performance differed between the two conditions. We found that they did not differ. More detailed results are below. For our analyses, one subject in each condition was removed for not responding during the experiment, and two subjects were removed from the Critiquing group for not completing at least 720 time steps of training (as did the remaining 57 subjects). The data from the teaching group—coming from the typical TAMER training scenario—is used as the second Tetris experiment in Chapter 3, where it is examined from a different perspective.

Plots of feedback frequency and performance by condition are respectively shown in Figures 5.2 and 5.3. A 2 (condition) x 9 (interval) repeated measures ANOVA indicated no significant main effect of condition nor an interaction of interval and condition for the dependent measure of frequency of responding (all $F[2,55] < 0.83$, $p > 0.60$). Considering agent performance (i.e., lines cleared by the simulated TAMER agent), there was no significant main effect of condition nor an interaction of interval and condition (all $F[2,55] < 1.14$, $p > 0.33$).

Seeking to assess how similar the effects of the two conditions are, we calculated a Bayes factor for the data. A Bayes factor is the odds that the null hypothesis is true when the alternative is a distribution over alternative hypotheses. We examined performance at the end of the nine intervals, giving something akin to a final skill level, and feedback frequency over all intervals. Using an effect-size scaled parameter of 1 for specifying the distribution of alternate hypotheses, we calculate the JZS Bayes factor to be 4.28 for performance and 4.67 for feedback frequency [84].

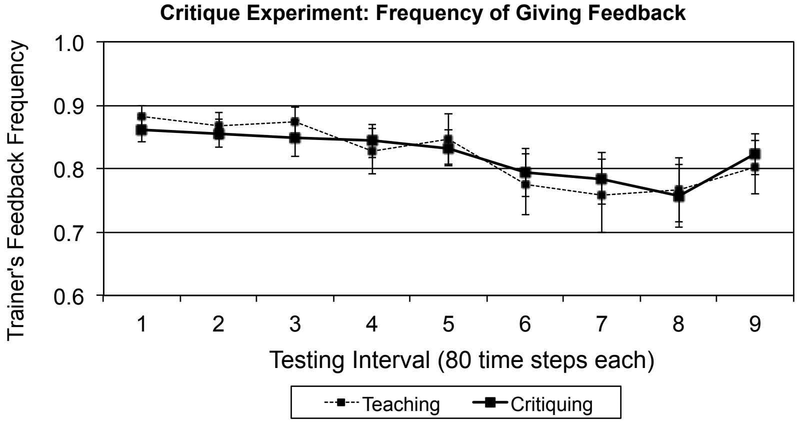


Figure 5.2: Feedback frequency from the human trainer over 9 bins of 80 consecutive time steps each. On all plots with error bars, the bars show standard error.

Thus, under this parameter—which is recommended as a default parameter because it favors neither outcome—the null hypotheses for both metrics is more than four times more probable than the alternatives, given the data. A Bayes factor above 3 is commonly considered substantial evidence that the null hypotheses is approximately true, giving us confidence to conclude that the subjects' roles had similar effects.

The similarity in outcomes for each condition might reflect the strength of the manipulation's effect. After training, we asked subjects, "If you felt your feedback had an effect on the computer, how much of an effect do you feel that it had?" and had them rate from 0–6 on a Likert scale, with higher values indicating that the feedback seemed to have more effect. Unfortunately, in hindsight we realized that this question is flawed; the use of word "feedback" instead of the more neutral "evaluation" that was used in the critique condition's instructions (see Appendix B) as well as simply raising the prospect of the subject's "effect" both likely undermine what belief had previously been held during training, pushing the responses to higher values. The teaching condition gave marginally higher ratings: a mean of 4.79 as

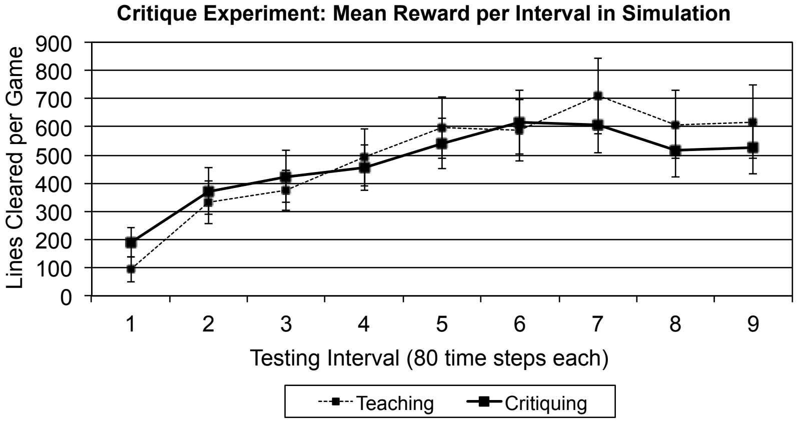


Figure 5.3: Performance (lines cleared per game) of policies fixed at the end of 9 intervals, each 80 time steps in length. In other words, the tested agents have learned from the first 80, 160, ..., and 720 time steps.

opposed to 4.31 for the critique condition. This difference is not statistically significant. Given that the manipulation check itself appears to bias responses towards similar values and that the actual responses are similar, we cannot deduce whether the manipulation was successful.

Though we are not focusing on the difference in the amounts of positive and negative reward given, I report that the mean absolute value of positive reward per time step was greater than that of negative reward across all conditions of both experiments (all $p < 0.025$ in paired t-tests). This finding confirms observations by Thomaz and Breazeal [105].

In summary, the difference in participants' roles did not significantly affect any of the the dependent variables. Looking at performance, a Bayes factor analysis suggests that similarity between the two groups can explain the lack of significance, as opposed to merely too few subjects or too high of variance.

This critique experiment influenced the following experiment on feedback-frequency in several critical ways. First, because teaching and critiquing trainers behaved and performed similarly, all conditions in the feedback-frequency experiment involve a teaching role for the subject. Second, because subjects' frequency of responding was quite high in the critique experiment, we changed the subjects' instructions from "If it has made a [good/bad] move, press ..." to "If you feel it is necessary to [reward/punish] it, press ...". From this change in instructions, we hoped to both lower their baseline frequency and give subjects more leeway to determine their own frequency, two consequences that we expected to increase any differences in frequency created by the different conditions. Lastly, after the conditions of this critique experiment did not significantly affect the rate of feedback that some authors predicted would improve performance, we were motivated to more directly manipulate feedback frequency by making the agent react to it.

5.2.2 Feedback-frequency experiment: varying action greediness with feedback frequency

In this section, I describe the feedback-frequency experiment, which investigates a human-agent interaction scenario in which the computer agent reacts to waning human feedback by behaving worse. By controlling the parameters of the computer agent's reaction to its human trainer's frequency of feedback, we were able to evaluate the human behavioral response under three conditions. The specification of conditions below relies on the term *greedy*, which in this context means choosing the action a that maximizes a prediction of immediate human reward, $argmax_a[\hat{R}_H(s,a)]$. To be concise and ease reading, I sometimes refer to non-greedy actions as "misbehavior", since agents are taking actions that they currently believe to be suboptimal (though they may actually be optimal).

Design

Subjects were randomly assigned to one of three conditions: Reactive Non-greedy, Greedy, or Yoked Non-greedy.

1. Greedy (n = 19): The TAMER agent always chooses the action with the highest predicted feedback value.[3]

2. Reactive Non-greedy (n = 30): The TAMER agent's level of greediness is negatively correlated with the recency-weighted frequency of human feedback. (The frequency is "recency-weighted" because more recent opportunities for feedback are weighted more heavily in the frequency calculation.) For this group and the Yoked Non-greedy group, details about calculating feedback frequency and its effect on action selection are described below in this section.

3. Yoked Non-greedy (n = 30): To separate the effects of general misbehavior from misbehavior that occurs in response to the trainer, we added a third group in which agents explore without being tied to their respective trainers. In this Yoked Non-greedy group, the TAMER agent uses the frequency from a matched trainer from the Reactive Non-greedy group instead of its own trainer's feedback frequency. In other words, we assign each member of this group to a member of the Reactive Non-greedy group. The agent explores based on feedback frequency, identically to the Reactive Non-greedy group, except that the frequency at step i is determined from the feedback history of the matched subject from the Reactive Non-greedy group rather than the current subject's feedback history. Thus, whereas the agent acts with varying degrees of misbehavior, the level of misbehavior was not causally determined by the subject's behavior.

We hypothesized that the Reactive Non-greedy group would have the highest feedback frequency as well as the best performance. Our intuition was that, in line with the idiom "the squeaky wheel gets the grease" and popular wisdom that

[3] The Greedy group can be considered similar to the Teaching group from the critique experiment. The two groups' instructions do contain differences, but both groups have identical TAMER agent algorithms and subjects are aware that they are teaching.

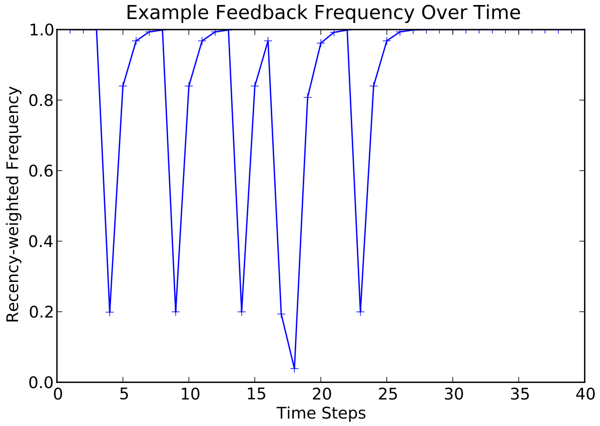


Figure 5.4: An example trajectory of recency-weighted frequency over the first 40 time steps of training. The frequency varies dramatically, a consequence of the small decay factor, 0.2. In this example, the trainer refrains from giving feedback only six times.

misbehavior can be a cry for attention, an agent that "misbehaves" when feedback frequency diminishes will be effectively training the trainer to give more feedback. And given more feedback, the agent would have more training samples to learn from, resulting in better task performance.

Calculating frequency To calculate a trainer's recency-weighted feedback frequency, each feedback instance is exponentially decayed over time. Thus, at each time step, we calculate $a := [decay * a] + (feedback \neq 0)$ and $b := [decay * b] + 1$, where a and b are initialized to zero and $feedback \neq 0$ resolves to 1 when feedback was given and 0 otherwise. Together, a and b define frequency: $freq := a/b$. In our experiments, the decay parameter was 0.2, which heavily weights the last few actions. An example frequency trajectory can be seen in Figure 5.4.

Choosing actions based on frequency Given a frequency, the agents in both non-greedy conditions choose actions. To choose, an agent ranks all available actions according to their predicted human reward, $\hat{R}_H(s, a)$, and picks out five actions from that ranking: the best, the second-best, the action at the first quartile,

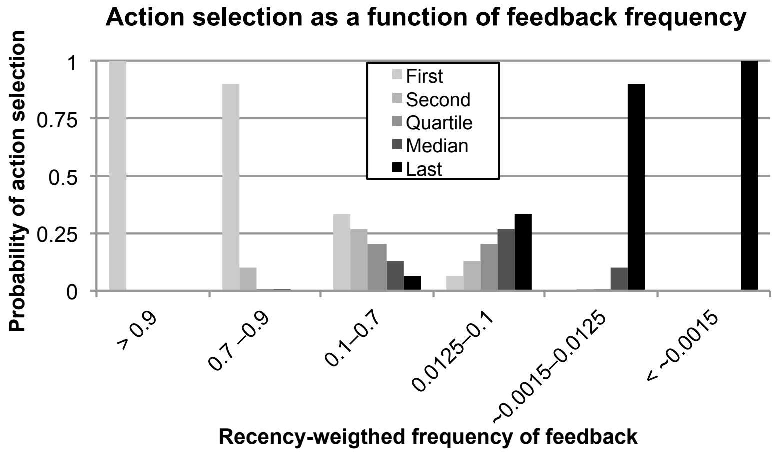


Figure 5.5: Probability distributions over the five possible actions at different recency-weighted feedback frequencies.

the action at the median, and the worst. (Ambiguous quartile and median choices go to the better-ranked action.) Then, the agent chooses randomly from these five actions according to a probability distribution conditioned on frequency, where lower frequencies generally result in worse-ranked action choices. The distributions can be seen in Figure 5.5.

Results

We performed the same descriptive and model-based analyses as we did for the previous critique experiment. Specifically, performance is again tested offline, not during training, and *the testing policy is greedy regardless of condition.* However, we find significant results here and thus do not perform the Bayes factor calculation, which we used to determine how similar the data was between conditions after finding a complete lack of significance. One Greedy subject, three Reactive Non-greedy subjects, and five Yoked Non-greedy subjects were removed for responding insignificantly during the experiment. Also, one Greedy subject, one Reactive Non-greedy subject, and four Yoked Non-greedy subjects were removed for training for less than the 800 time steps we used for analysis. For the two non-greedy conditions, subjects matched to removed subjects were also removed from analysis.

If non-greedy actions increase feedback frequency and tying non-greedy actions to trainer's recent feedback frequency further increases subsequent frequency, we expect the Reactive Non-greedy group to have the highest frequency, followed by the Yoked Non-greedy group, with the Greedy group having the lowest frequency. And since frequency increases the number of learning samples, we expect the same ordering of performance.

Trainers' feedback frequencies are shown in Figure 5.6, and performance after each training interval is shown in Figure 5.7. Note that the change in instructions described at the end of Section 5.2.1 was effective: the baseline feedback frequency, given by the Greedy group, is lower than the almost equivalent Teaching group in the critique experiment.

Surprisingly, 2 (condition) x 10 (interval) ANOVAs comparing the performance (i.e., lines cleared) of the Greedy group over all intervals to that of the Reactive Non-greedy and Yoked Non-greedy groups found significant effects by condition ($p = 0.015$ and $p = 0.024$, respectively), indicating superior learned performance within the greedy group. The two non-greedy groups were not significantly different. Also, 2 x 10 ANOVAs comparing trainer's feedback frequencies found no significant differences.

Results for good trainers only Before acting intelligently, these learning agents go through a period of initial learning, during which their actions are generally of low quality. Additionally, many agents are never trained to a level at which greedy actions are generally good. Taking non-greedy actions when greedy actions themselves are not good lacks the qualitative characteristic on which we are focused: non-greedy action corresponding to decreased quality of action. Therefore, we repeat

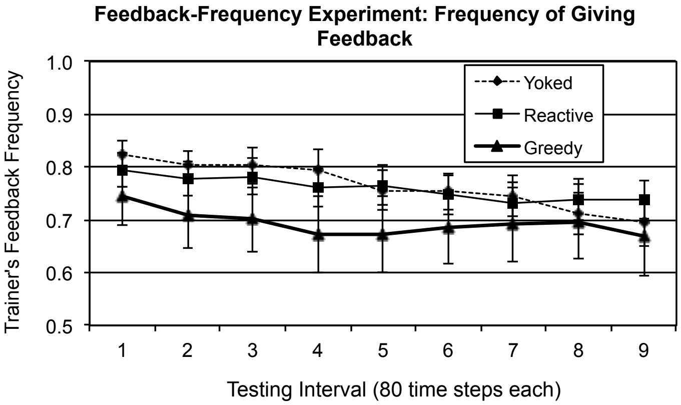


Figure 5.6: Feedback frequency from the human trainer over 9 bins of 80 consecutive time steps each.

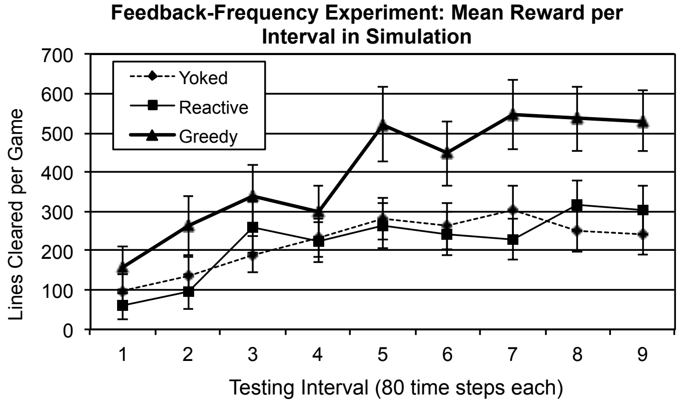


Figure 5.7: Performance (lines cleared per game) of policies fixed at the end of 9 intervals, each 80 time steps in length.

the analyses above, only examining the subset of subjects who were able to train their agents to consistently clear more than 10 lines on average across multiple time intervals. Additionally, we only use data starting at the third interval, where the percentage of agents that pass the 10-line standard first surpasses 90% (after pass rates of only 58.3% and 72.2% in the first two intervals), never dropping below after. The 10-line threshold was chosen for its position in the valley of the bimodal distribution of agent performance across subjects.[4] This more selective analysis gives a different perspective that is more focused on the effect of "misbehaving" to affect the trainer.

After removing low-performing subjects and all subjects that were matched to those low-performing subjects, the condition sizes were Reactive Non-greedy, n = 10; Greedy, n = 16; and Yoked Non-greedy, n = 10. The feedback frequency by condition across time intervals for this smaller set of subjects is shown in Figure 5.8. Compared to the full set of subjects, all conditions have generally higher feedback frequencies. However, this frequency increase is more pronounced in the two non-greedy conditions. Further, the Reactive Non-greedy condition now results in more frequent feedback than the Yoked Non-greedy condition. Despite the small number of subjects being considered, the Reactive Non-greedy group's mean feedback frequency over intervals 3-10 has marginal significance ($p < 0.1$) in comparison to the lower feedback frequency of the Greedy group. Additionally, a non-parametric analysis of the same data is nearly significant, with an upper confidence bound of 0.83 for the Greedy group and a lower confidence bound of 0.82 for the Reactive Non-greedy group. Therefore, we expect that increasing the number of subjects would quickly strengthen the significance of the difference in feedback frequency.

The performance of these subjects across conditions is much more similar than in the full set of subjects and is no longer significantly different. However, removing subjects based on performance clearly biases group performance. This bias is motivated for analyzing feedback frequency but not for performance, and we therefore only base our performance results on the full set of subjects. We can say, though, that these results add further evidence (though biased) that increased feedback frequency and the consequently increased number of learning samples do

[4] Illustrating the bimodality of performance, there were 79 subjects across conditions. In the 9th testing interval, 23 agents clear between 0–1 lines; 47 clear more than 100. Only 2 agents clear 5–20 lines.

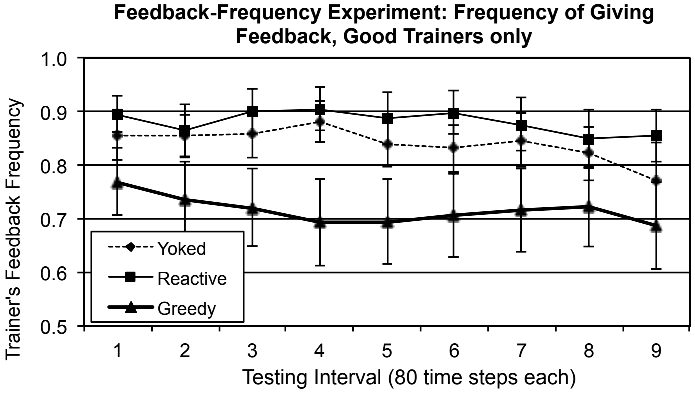


Figure 5.8: Figure 5.6 with low-performing trainers removed.

not result in better performance in this experiment.

5.3 Discussion

In this section, the results of the experiments in Section 5.2 are interpreted and discussed. The first subsection draws a conclusion on agent design from the critique experiment; the second subsection does likewise from the feedback-frequency experiment. I then discuss the feedback-frequency experiment's implications for the explore-exploit dichotomy that is pervasive within the field of learning agents. Lastly, I discuss the technique of using social agents to study human behavior, using our experiments as examples and describing how these results may also be of interest outside of artificial intelligence communities.

5.3.1 Honesty is still the best policy

When agents learn to perform tasks, one clear objective is to maximize performance. The results from the critique experiment indicate that, contrary to the hypothesis that human trainers would need to be deceptively told that they are not teaching to do their best training, the human-agent system performs similarly when the human knows that he is engaged in a training session. Either the subject's given role had little or no effect on his feedback, or the TAMER agent was unaffected by these differences in feedback.

In addition to the performance objective that we explicitly study, it is also important to respect the desires and needs of humans. Deceiving human trainers to get the best performance is an ethically questionable trade-off. The results provide evidence that disclosing to the trainer that he is teaching maximizes both crucial objectives, performance-based and humanistic.

5.3.2 A tool for increasing feedback

When numeric feedback comes to an agent from an encoded reward function instead of a human, the problem is often framed as a reinforcement learning problem. These problems are usually formalized as Markov Decision Processes (MDPs). In an MDP, reward has a static distribution of frequency and quality. In contrast, human reward can be affected along both of these dimensions. From this observation, one may notice that one way to give highly effective feedback (though possibly imperfect feedback with certain function approximators) for a TAMER agent would be to give feedback at every time step and have as its value the expected return of MDP reward under the optimal policy from the most recent state-action pair, where the MDP reward follows the task's objective and credit is assigned only to the preceding time step. These two characteristics of feedback—frequency and quality, or, equivalently, the number of learning samples and the quality of their labeling—comprise two dimensions along which a particular human trainer's feedback can be improved.

The feedback-frequency experiment demonstrates one on-line technique for increasing the frequency of human feedback: lowering action quality. More specifically, when examining only the successful trainers (for which non-greedy actions would actually look worse), tying the action quality to the trainer's recency-weighted frequency further increased feedback frequency. Considering that there are likely other techniques that increase either frequency or quality of feedback, one product of our results is a proof-of-concept that this broader category of techniques exists, though the extent of its breadth is an open question. Also, the concept of an agent manipulating the trainer to improve feedback can be generalized to other modes of teaching, including demonstrations, which can also vary by frequency and quality.

Contrary to our expectations, though the agents' manipulations increased feedback frequency, they did not improve performance and even decreased it among the full set of subjects. Exploring created more learning samples, but we suspect these samples were less useful than those experienced by the Greedy group. We see two plausible explanations: the learning samples were in a less useful area of the state-action space, or the quality of trainer feedback worsened. The intuition behind the first potential explanation is that the learning samples created during greedy behavior help distinguish between the best few actions, whereas non-greedy behavior created samples that help distinguish between mediocre or worse actions, a type of differentiation that does not aid an agent trying to choose the best action. Further, the samples from non-greedy actions may have even been directly harmful; the representation of $\hat{R}_H$ is not highly expressive, and more accurately modeling reward for non-greedy actions likely lessens the accuracy of modeling high-quality actions. The other potential explanation is that the quality of the feedback within the non-greedy conditions suffered because of trainer frustration or some other effect of misbehavior on the trainer. Further analysis of the data might shed light on which of these explanations is correct. For instance, we could test each agent's performance with the same learning samples, except we label each sample with a static feedback function instead of with the variable set of humans that did label the samples. This relabeling would control for quality of feedback, directly testing how much the difference in the samples' locations in state-action space would affect performance.

More generally, whether misbehavior can be used to increase interaction *and* learned performance is a promising question for future inquiry.

The agent's frequency-tied action selection can least speculatively be framed as a "manipulation". We might also consider it to be a form of communication with the human trainer, though we are careful not to imply that the trainer consciously understood some message from the frequency-tied actions, which she may or may not have. Another speculative but plausible interpretation is that when the agent lowers its action quality after the trainer's feedback drops in frequency, the human is being punished for inattentiveness. This interpretation is more compelling if the human trainer is emotionally vested in the agent's performance, which fits anecdotally with comments made by subjects and the authors' experience in informally training agents themselves.

One lesson of this feedback-frequency experiment is that agent designers should be careful not to make the mistake of considering pedagogy to be a single-directional manipulation and assuming that though teacher and student do interact, it is the student who is significantly changed through the interaction. On the contrary, the student has expectations of the teacher and beliefs about how the teacher should meet his or her needs, and an effective student will teach the teacher how to meet those needs. Relatedly, Breazeal has argued that the teaching of agents should be considered a "tightly coupled joint action" in which both human and computational agent collaborate and adapt [15].

5.3.3 Non-greedy action is not necessarily exploration

When referring to agents that learn to estimate some notion of the relative values of various state-action pairs (i.e., not policy-search learners), researchers generally consider actions to be either exploratory or exploitative. This dichotomy between exploration and exploitation holds strictly in traditional reinforcement learning, where an action a is exploitative if it is chosen greedily, such that ($a = argmax_a Q(s,a)$), and contrapositively any action chosen non-greedily, typically resulting in $a \neq argmax_a Q(s,a)$, is exploratory [90].[5]

At the intersection of learning agents and human-agent interaction are agents that, like TAMER agents, learn interactively from human teachers. In past work, many of these agents only exploit [47, 3, 72] and some, especially those that use reinforcement learning, explore or exploit [105, 36, 116]. However, I will argue that the non-greedy actions taken by agents in the Reactive Non-greedy group of the feedback-frequency experiment are neither exploration nor exploitation.

Is it exploitation?

Retaining the notion that exploitation involves greedy action selection, the Reactive Non-greedy group's non-greedy behavior was not exploitation by definition. This conclusion generalizes to any agents that learn the values of state-action pairs for the task and cannot model the human as part of their value function, though they may be able to model the impact of their actions on the trainer's feedback frequency and quality.

Is it exploration?

In reinforcement learning terminology, any action a such that $a \neq argmax_a \hat{R}_H(s,a)$ is commonly referred as "exploration". But exploration in reinforcement learning, and in general if we want to keep the term close to its colloquial meaning, is undertaken to learn more about state-action pairs which are not experienced sufficiently during greedy behavior to create the desired level of behavioral improvement. The

[5] Though exploration is often considered equivalent to non-greedy action, this definition does not fit all instances of its use in RL. For instance, an agent that employs an exploratory policy might have a greedy policy that sometimes agrees on what action to select. However, this is a semantic point that does not affect our assertion that the comprehensive dichotomy of explore/exploit is insufficient.

Reactive Non-greedy group in the feedback-frequency experiment may have received a wider range of state-action pairs in their learning samples as a result of their non-greedy behavior, but they also affected their feedback source. Their trainers' feedback frequency, on average, was higher than that of other groups, sometimes significantly so, giving the agents motivation beyond exploration to act non-greedily.

Through its non-greedy actions, an agent in the Reactive Non-greedy group does receive information about state-action pairs that it would likely not encounter during greedy actions. So, in a sense, exploration does occur. But exploration is not the only effect, and in an agent that predicts the effects of its actions and acts with goals, the exploration may be merely incidental to the intended result of increasing feedback frequency. Thus, while the agents' non-greedy actions had exploratory consequences, calling such actions exploration is incomplete, obscuring their desirable, non-exploratory effects.

There is more than one reason to act non-greedily. Exploring is one reason, as is increasing a trainer's feedback frequency. If the learning agents community intends to embrace the use of humans as teachers for agents, it might reconsider the common practice of using the word "exploration" synonymously with non-greedy actions. Though exploration remains a critical form of non-greedy action, our results show that when a human trainer is in the learning loop, there are reasons to act non-greedily besides exploration.

5.3.4 Illustration of employing human-agent interaction to study human behavior

In this subsection, I conduct a more general discussion on the merits of using social robots or social software agents to study human behavior outside of human-agent interaction. Our experiments serve as motivating examples in this discussion.

Computational agents, both robotic and simulated, comprise an emerging tool for the behavioral sciences. In current practice for experiments on human behavior that require social interaction and constrained behavior on one side of the interaction, a human fulfills the role opposite the subject. Compared to this human actor,[6] a computational agent can act more consistently, since its behavior is fully parametrized. Further, the conditions under which humans act may confound their performance. In our feedback-frequency experiment, for example, a human pupil's learning would likely be confounded by varying levels of mental effort to align actions to the constraints of each condition. The computational agent chooses its actions without meaningfully pulling resources from the learning algorithm (i.e., though they share computation time, there was plenty of time for both learning and action selection). Additionally, the computational agent can record every aspect of its "mental" process and behavior, allowing in-depth analysis later. Both experiments provide an example of such analysis, freezing learning at different points in time and testing performance. On the other hand, human actors have some clear advantages. The focus of studies on social interaction is generally human-human interaction, and human subjects probably interact more naturally with human actors than computational ones, though the extent of this difference will depend on the character of the computational agent. Thus, the relative generalizability of results from experiments with human actors increases from the authenticity of human-human interaction. Given the different strengths of human and computational agents, we expect both to play an important role in future behavioral studies, a view shared by many in the human-robot interaction community [69, 14, 62, 22].

This chapter provides analysis aiming to be valuable to a researcher of learning agents or human-robot interaction. However, these results may also be of interest

[6] A human opposite the subject could have fully scripted behavior, act naturally except in certain situations (like misbehaving at certain times), or simply act naturally. Additionally, the subject may believe either that this person is a fellow subject or that she is working for the experimenters. I call this human that would potentially be replaced by an agent a "human actor" for simplicity and to differentiate from the subject.

to the educational community. There the relationship between classroom misbehavior and teacher attention is of real-world importance [115]. In a relatively recent article, Dobbs et. al [23], summarizing past research on the relationship between misbehavior and attention from teachers, write that “children who misbehave frequently receive more teacher attention than do children who rarely misbehave.” One study found that the amounts of criticism and commands received from a teacher were negatively correlated with the level of on-task behavior from children [26]. Other research on this relationship has been correlational and often considers a potential causal relationship in the direction of attention causing misbehavior. Using real children as misbehaving confederates in a randomized controlled trial is an untenable proposition. But with interactive agents, we were able to establish the first causal connection between misbehavior and teacher attention, showing that performance-oriented misbehavior can increase attention.

5.4 Summary

Unlike the other core chapters of this thesis (3, 4, 6), this chapter focuses more on the human trainer than on the learning algorithm. In this chapter, I describe two experiments that consider how human beliefs and agent behavior affect a human’s teaching. The first, the critique experiment, showed similar feedback frequency and agent performance between subjects placed in a teaching role and subjects in a critiquing role, indicating that either the given role had little effect on the subject’s feedback or it did affect feedback but the resultant differences did not affect the TAMER agent’s performance. The second, the feedback-frequency experiment, demonstrated a technique that agents can use to increase the frequency of trainer feedback: acting non-greedily. Additionally, when we filter for agents that show sustained decent or better performance, the frequency increase is greatest when this non-greedy misbehavior occurs in response to decreases in the trainer’s feedback rate. Through this type of behavior, the feedback-frequency experiment also gives a specific example of how actions in the presence of a human trainer can be used for purposes other than exploration or exploitation. This result shows that the explore/exploit dichotomy is inadequate for describing actions by an agent learning interactively from a human. Together, these experiments 1) lend support to the efficacy of the TAMER approach—actively taught and thus far greedy—to learning from human reward and punishment, and 2) identify forms of human-agent interactivity that do or do not impact agent performance.

This research may serve as a model for other research that studies humans by having them interact with robots. The generality of our findings would be buttressed by repeating these two experiments in different contexts: especially using a robotic agent, different tasks, and even a different teaching modality, such as learning from demonstration. Nonetheless, the results presented here provide interesting, sometimes surprising results that apply to designers of learning agents, including social robots. And the unexpectedness of some of our conclusions indicates that further studies of human teaching stand to provide much counterintuitive guidance in the design of agents that learn from human teachers.

An agent with the power to manipulate the trainer to its advantage should not necessarily use that power. We should consider when pulling a teacher in for more training is worth the cost in human effort. There are numerous potential approaches to this problem. For example, a more sophisticated agent might have some self-confidence measure and only engage the human when it lacks confidence in making decisions [18].

Lastly, this chapter's two experiments serve as exemplars of using *learning* agents as parametrized social entities in experiments on human behavior. I hope that they will inspire and guide researchers to explore this nascent experimental technique, helping to expand the impact of human-agent and human-robot interaction into the behavioral sciences.

This chapter and the previous build upon Chapter 3's introduction of the TAMER framework but do not directly address the fundamental question of how to solve the interactive shaping problem. In the following chapter, I reexamine this problem and rigorously investigate alternative solutions to it. Ultimately, the TAMER approach is supported under certain more obvious assumptions, but the data also point in a potentially more powerful direction in which all tasks are formulated as continuing.

Chapter 6

Discounting Human Reward: Limitations of Episodicity

Several studies, including those in Chapters 3 and 4 of this thesis, have demonstrated that human-generated reward can be a powerful feedback signal for control-learning algorithms. However, the algorithmic space for learning from human reward has hitherto not been explored systematically. Using model-based reinforcement learning from human reward in goal-based, episodic tasks, this chapter investigates how anticipated future rewards should be discounted to create behavior that performs well on the task that the human trainer intends to teach. We identify a "positive circuits" problem with low discounting (i.e., high discount factors) for episodic tasks that arises from an observed bias among humans towards giving positive reward. Empirical analysis verifies the existence of the positive circuits problem and further indicates that high discounting (i.e., low discount factors) of human reward is necessary in goal-based, episodic tasks. Lastly, an alternate strategy for overcoming the positive circuits problem—converting the episodic task to a continuing one—is shown to support a wide range of discounting and therefore provide greater algorithmic flexibility. In the concluding summary of this chapter, I discuss the impact of

these results on best practices for learning from human reward.

Social rewards and punishments powerfully influence animal behavior, humans included. In recent years, this form of communication has been adapted to permit teaching of artificial agents by their human users, both in work described in earlier chapters of this thesis and elsewhere [36, 107, 101, 88, 76], which I describe in Section 2.5. This form of teaching is formally defined as interactive shaping in Chapter 2. In short, "human reward" is conceptually communicated to the trainer as signaling degrees of reward and punishment, approval and disapproval, or something similar, and the reward is received by the learning agent as a scalar value through varying interfaces (e.g., keyboard, mouse, or verbal feedback).

In Chapter 1, I assert that interactive shaping enables people—without programming skills or complicated instruction—(1) to specify desired behavior, and (2) to share task knowledge when correct behavior is already indirectly specified (e.g., by a pre-coded reward function). The first of these functions of interactive shaping is the focus of Chapter 3, and Chapter 4 explores the second. Further, in contrast to the complementary approach of learning from demonstration [4], learning from human reward employs a simple task-independent interface, exhibits learned behavior *during* teaching, and, we speculate, requires less task expertise and places less cognitive load on the trainer. Section 2.5.3 elaborates further on learning from demonstration and its comparison to interactive shaping.

This research in this chapter is the first to assess a fundamental aspect of interactive shaping: how expectations of future human reward are discounted when an agent evaluates the quality of available actions. As we detail in Section 6.1, past work on learning from human reward has consistently employed relatively high discount rates (some of which were ascertained through email with the authors). This trend has gone unnoticed until now; this chapter both identifies and justifies the trend. Besides being a curious aspect of past work, the question of discounting human reward is crucial because discounting directly determines what learning algorithms can be used and the flexibility of the agent (discussed in Section 6.2). Additionally, the comparative analysis within this chapter gives structure to the body of past work on learning from human reward, which previously lacked comparison between studies.

Recall from Section 2.1 that reinforcement learning without hidden states usually concerns solving tasks formulated as Markov decision processes (MDPs), denoted as $\{S, A, T, D, R, \gamma, \}$. RL algorithms seek to learn policies ($\pi : S \to A$) for an MDP that maximize return from each state-action pair, $Q^\pi(s, a)$, where $Q^\pi(s, a) = \sum_{t=0}^{\infty} E_\pi[\gamma^t R(s_t, a_t)]$. I will refer to such return-maximizing policies as *MDP-optimal*.

In this chapter, we do not focus on designing algorithms that achieve MDP-optimal behavior. Rather, we investigate how to define an MDP such that MDP-optimal behavior performs well on the task the trainer intends to teach, as measured by a task performance metric τ. This problem is challenging because we, as algorithm designers, cannot specify the reward function; we leave that specification to the human trainer. But the discount factor can be controlled, as can the agent's perception of transitions between states; we investigate the effect of various parameter settings in our experiments.

We specify MDPs in which the reward function is a predictive model of human reward, $\hat{R}_H : S \times A \to \mathbb{R}$, creating an MDP $\{S, A, T, \hat{R}_H, \gamma, D\}$.[1] If an agent knows such an MDP, it can find the MDP-optimal policy, but that policy is *not* guaranteed to be the best possible policy according to τ. Indeed, for common reward functions, each choice of γ will lead to a *different* MDP-optimal policy. We

[1] Thus, the MDP changes any time the human reward model $\hat{R}_H$ is modified by further training experience.

seek the setting of γ that leads to the best MDP-optimal policy according to τ.[2] Note that this problem specification is narrower than the full interactive shaping problem from Section 2.3—which says nothing about MDPs—but is a generalization of the TAMER approach from Chapter 3, since TAMER hypothesizes a process for learning $\hat{R}_H$ and then essentially behaves as if it is solving the corresponding $\hat{R}_H$-inserted MDP using $\gamma = 0$.

This chapter presents an application of model-based RL to learning from human reward (though this contribution is not our focus), where the reward function is learned from a human trainer and the transition function may be given, as it is in our experiments, and the agent plans with the two models. We find the model-based approach more informative than model-free RL because giving the agent knowledge of the MDP specification allows an agent to learn policies that perform well on the MDP more quickly, making agent behavior more effectively reflect the current $\hat{R}_H$, approaching and often achieving MDP-optimal behavior that allows evaluation of the MDP specification itself.

In MDPs, future reward is sometimes discounted to make reward in the near future more valuable than later reward. With our currently limited knowledge of the properties of human reward, it is unclear whether and to what degree human reward should be similarly discounted to create the behavior desired by the trainer. Thus, we ask the following experimental question: what discount factor maximizes the MDP-optimal policy's task performance, measured by τ?

We focus on *episodic tasks* [90] that are goal-based, meaning that the agent's task is to reach one or more goal states, after which the learning episode ends, a new episode starts with state chosen independently of the reached goal state, and the agent experiences reward that is not attributable to behavior during the previous

[2] Though τ may be subjective and flexibly defined in practice, in this chapter the trainer is given a static, pre-specified τ to maximize, facilitating empirical evaluation of the MDP's resultant task performance.

episode. As we explore throughout this chapter, goal-based tasks have characteristics that make a comparison of different discounting rates especially informative. Despite our focus on goal-based tasks, however, we seek an algorithm that effectively learns in all episodic tasks, whether goal-based or not.

I briefly review past work in Section 6.1 and discuss the consequences of the two extreme rates of discounting, $\gamma = 0$ and $\gamma = 1$, in Section 6.2. Note that $\gamma = 0$ is the approach taken by the TAMER framework. Section 6.3 presents a hypothesis about discounting in episodic tasks—that *maximizing only immediate reward, i.e.,* $\gamma = 0$*, results in the best task performance*—and an intuitive argument for the hypothesis' likelihood that is built on observations that humans tend to give more positive reward than negative reward, creating what we term the *positive circuits problem.* In Section 6.4, I describe two empirical analyses of discounting that support our hypothesis and the intuition behind it. Section 6.5 presents an alternative approach to overcoming the positive circuits problem, *modifying the agent's perception to make the episodic task appear continuing.* We show that this modification does indeed overcome the positive circuits problem, creating roughly equivalent performance for all discount factors $\gamma < 1$, and then demonstrate the potentially powerful benefits of using higher discount factors. Through this analysis, *we achieve the first reported instance of an algorithm learning successfully from human reward using low discounting.* Altogether, the results of this chapter provide strong support for the TAMER framework in episodic domains yet point towards more powerful but algorithmically challenging approaches that discount future reward less under the assumption that tasks are always formulated as continuing.

6.1 Discounting in past work on learning tasks from human reward

In Section 2.5.1, I review past work on learning tasks from human reward. Curiously, all previous algorithms have discounted more severely than is typical for MDPs. For episodic tasks, researchers have discounted by $\gamma = 0.75$ [107] and $\gamma = 0.9$ [101]. In continuing domains, $\gamma = 0.7$ [36], $\gamma = 0.75$ [88], $\gamma = 0.9$ [60], and $\gamma = 0.99$ [76] have been used.[3] The $\gamma = 0.99$ work is a non-obvious example of high discounting; with time steps of 5 ms, reward one second ahead is discounted by a factor of approximately 0.134. At the extreme of this trend, the TAMER framework discounts by $\gamma = 0$, learning a model of human reward that is (because of this discounting) also an action-value function. This pattern of myopic maximization of human reward has hitherto not been identified.

In many of these studies, including those on TAMER, learning from human reward is shown to improve in some respect over learning only from MDP reward (sometimes the championed learning algorithm uses both human and MDP reward and sometimes also a form of action suggestions) [107, 101]. In most of the others, learning from human reward is shown to be effective in a task where specifying an MDP reward function would be infeasible in the motivating use case [36, 76] (i.e., training a user-specific policy when the user cannot program).

6.2 Consequences of discounting

The two extremes of discounting have different advantages, briefly described in this section.

For $\gamma = 1$, the agent acts to maximize the undiscounted sum of future re-

[3] The discount factors for three publications were learned through personal correspondence with an authors Isbell [36] and Morales [101, 60].

ward. With this discounting, the reward function could encode a trainer's desired policy, the trainer's idea of the task goal, or some mixture of the two; expression of a task goal permits simpler reward functions (e.g., 0 for transitions that reach goal state and -1 otherwise), which could reduce the need for training, allow the agent to find behaviors that are more effective than those known by the trainer, and make the agent's learned behavior robust to environment changes that render ineffective a previously effective policy but leave the purpose of the task unchanged (e.g., when the MDP-optimal path to a goal becomes blocked, but the goal remains unchanged). Given a model of system dynamics (i.e., a transition model) and a planning algorithm, these advantages become even more pronounced.

For $\gamma = 0$, the agent acts myopically to maximize immediate reward. This objective is simpler algorithmically, since a discount factor of zero reduces reinforcement learning to supervised learning. Supervised learning is generally an easier problem, and such discounting enables the agent to build upon a larger body of past research than exists for reinforcement learning, including tools for automatic selection of features, the representation of the human reward model, and the algorithm for learning parameters of this model. A disadvantage of this discounting, on the other hand, is that the reward model can encode a policy but not more general goals of the task.

Our ambition in this work is to create a natural interface for which people generate reward on their own. Accordingly, we observe that *algorithm designers should choose a discounting level that is compatible with human reward rather than assuming the human trainers will fit their reward to whatever discounting is chosen.* Granted, there appears to be some flexibility in the choice of algorithm: trainers can be instructed before they teach, and humans appear to adapt to the interface and learning algorithm with which they interact. But it may nonetheless be the case that certain intuitively appealing algorithms are incompatible with some or all human training, even after instruction and practice. The rest of this chapter explores such a possibility.

6.3 The positive circuits problem of learning from human reward with high γs

In this section, I describe our intuition in two parts for why treating human reward identically to conventional MDP reward in episodic, goal-based tasks—i.e., using γ at or near 1—will often cause minimal task performance, a situation we call the positive circuits problem. In the discussion below, I assume MDP-optimal policies when referring to expected return.

6.3.1 Humans tend to give more positive than negative reward

Thomaz and Breazeal conducted experiments in which humans train agents in an episodic, goal-based task [107]. Focusing on the first quarter of the training session, when the agent's task performance is generally worst, they found that 16 out of 18 of their subjects gave more instances of positive reward than of negative reward. Over all subjects and the entire training session, 69.8% of reward instances were positive.

In Section 3.4.3, we also examine the balance of positive and negative reward, specifically from 27 subjects teaching TAMER agents to play Tetris and 19 subjects teaching TAMER agents to perform the mountain car task (as defined in Sutton and Barto [90]). Comparing the sums of each trainer's positive reward values and negative reward values, we find that 45 of the 46 trainers gave more positive reward than negative over their training session. The one exception, a mountain car trainer, gave an equal amount of positive and negative reward. The Tetris agents of eight trainers could not clear even 10 lines a game, in many cases averaging less than a line cleared per game. Yet these trainers still gave more positive reward than negative reward, despite deplorable task performance.

Based on past experiments, human trainers appear to generally give more positive reward than negative reward, often with remarkable consistency.

6.3.2 Consequences of positive reward bias for learning with large discount factors

In many goal-based tasks, there exist behavioral circuits that the agent can repeatedly execute, returning to the same or similar states. Such circuits exist for many MDPs, including any deterministically transitioning MDP with at least one recurrent state and any MDP that contains at least one state in which an agent can remain by taking some action. A simple example is an agent walking in circles in a navigational task. For such tasks, given the predominance of positive reward, it is likely that at least one such circuit will provoke positive net reward over each traversal of the circuit. Assuming that the goal-based task is episodic (i.e., a goal state is an absorbing state that ends a learning episode, a large class of problems), the MDP's discount factor γ is conventionally 1. Given that $\gamma = 1$, the expectation of return from states along a net-positive reward circuit will consequently be infinity, since the return is the sum of infinitely repeated positive reward. *Therefore, if a circuit exists with net-positive reward, an MDP-optimal policy for* $\gamma = 1$ *will never reach the goal*; an absorbing state ends accrual of reward, making the return of a goal-reaching state-action pair finite, regardless of how large the reward is for reaching the goal. Thus, we call this issue the positive circuits problem. The general problem of positive circuits in RL has been discussed previously [82, 70] but to my knowledge has not been connected to human-generated reward or episodicity.

The positive circuits problem is most clearly explained for the case when $\gamma = 1$, but circuits with net-positive reward can also be problematic at high γ

values that are less than 1. For instance, if $\gamma = 0.99$ and some circuit exists that has an average reward of 0.5 per transition, expected return from at least one state in this circuit will be approximately 50 or higher (because $\sum_{t=0}^{\infty}(0.99^t \times 0.5) = 50$). Though finite, such high expectations of return may, despite the trainer's best efforts, be larger than the expectation of return for any path from the state to the goal.

Trainer adaptation may be insufficient to avoid such a goal-averse result; delivering reward such that there are zero repeatable circuits of positive net reward may be severely unnatural for a trainer. Consequently, we hypothesize that RL algorithms using low γs will generally perform better on the trainer's internal task performance metric τ on goal-based, episodic tasks.

6.4 Empirical analysis: discounting in episodic tasks

In this section, I present two empirical analyses of the impact of different discount factors when learning goal-based, episodic tasks from human reward. Recall that, as discussed in Section 6.1, maximizing the discounted sum of human reward is not equivalent to maximizing task performance. In fact, it is precisely the relationship between these two types of maximizations that we are investigating.

In both analyses, the model of human reward, $\hat{R}_H$, is learned through the TAMER framework, and the output of this model provides reward for the agent within an MDP specified as $\{S, A, T, \hat{R}_H, \gamma, D\}$. During training, $\hat{R}_H$ is updated by human reward signals. The agent seeks to maximize the expectation of the sum of $\hat{R}_H$'s future output from any given state, $Q^{\pi}(s,a) = \sum_{t=0}^{\infty} E[\gamma^t \hat{R}_H(s_t, \pi(s_t))]$, but the agent is evaluated by a task performance metric τ. From a start state, the expected return of predicted human reward is denoted $V_{\pi}(s_o)$. For both tasks used below, the conventional MDP specifications (i.e., with hard-coded reward functions) have $\gamma = 1$; thus, at $\gamma = 1$ $\hat{R}_H$ is being used as if it were interchangeable with a conventional MDP reward function.

During training for both analyses, human reward was given via two keys on the keyboard, which mapped to 1 and -1. This mapping, though not infallible, is an intuitive choice that is similar to that of related works that explain their exact mappings [107, 101, 76, 88].

6.4.1 Varying γ with pre-trained human reward models

This first analysis uses 19 fixed $\hat{R}_H$s learned from the training logs created during the experiment described in Section 3.4.2, taken from the third run of 19 trainers of the mountain car task. Recall from Section 3.4.1 that in mountain car, a simulated car must accelerate back and forth across two hills to reach the top of one. We call this experiment the "γ-independent model experiment" because the human reward data was gathered under $\gamma = 0$ discounting, which differs from the discounting of most of our experimental conditions. I discuss possible training bias caused by such mismatched training and testing at the end of this section.

The RL algorithm is an enhanced SARSA(λ) algorithm that exhaustively searches a transition tree up to 3 steps ahead.[4] For these experiments, the agents learn from $\hat{R}_H$ for 4000 episodes, and episodes are terminated (with an update of 0

[4] This algorithm estimates return for each possible immediate action by taking the highest-return path on that action's branch, where a path's return is calculated as the sum of discounted reward along the path and the discounted, learned return at the leaf state of the path. Action selection is similar to ϵ-greedy: there is a probability ϵ at each step that the agent will choose a uniformly random action, and otherwise the action is that with the highest estimated return. Lastly, the depth of the agent's exhaustive tree search is chosen from a Uniform(0,3) distribution at each step to provide a wider range of experiences. The agent updates its value function only on experienced transitions. The SARSA(λ) parameters are below, following Sutton and Barto's notation [90]. The action-value function Q is represented by a linear model over Gaussian RBF features. For each action, 1600 RBF means are located on a 40×40 evenly spaced grid over the state space, where the outermost means in each dimension lie on the extremes of the dimension. Additionally, an activation feature of 0.1 is added for each action, creating a total of 4803 state-action features. When an action is input to Q, the features for all other actions are zero. The width σ^2 of the Gaussian RBFs is 0.08, following Sutton and Barto's definition of an RBF's "width" and where the unit is the distance in normalized state space between adjacent Gaussian means. All weights of Q are optimistically initialized to 0. The SARSA(λ) algorithm uses ϵ-greedy action selection, starting with $\epsilon = 0.1$ and annealing ϵ after each episode by a factor of 0.998. Eligibility traces were created as replacing traces with $\lambda = 0.84$. The step size $\alpha = 0.01$.

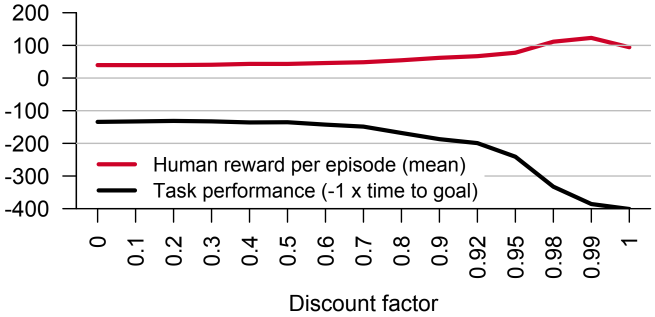


Figure 6.1: Aggregate results for the γ-independent model analysis, showing mean task performance and mean sum of $\hat{R}_H$-generated reward per episode for mountain car, over the final 500 episodes of 4000 episodes of learning.

reward) if the goal is not attained after 400 time steps, limiting the agent's maximum return to a finite value.

The $\hat{R}_H$s—the trainer models—were learned with the same linear representation over Gaussian RBF features that was used during the live training session, updating by incremental gradient descent. Each $\hat{R}_H$ trained on the first 20 episodes of its corresponding training log. To account for a small step size during gradient descent (0.001), each $\hat{R}_H$ was trained from 100 epochs on the trainer log. Credit assignment was performed by the delay-weighted, aggregate reward method from Section 3.3, updating only when reward was received as in the reward-only condition described in Section 3.4.3.

Figure 6.1 displays mean task performance and mean total reward per episode for each tested discount factor across all 19 $\hat{R}_H$ models. Additionally, Figure 6.2 displays the same data for each model separately to allow further inspection and to show the consistency of qualitative results between various models. We consider final performance to be over the last 500 episodes of learning.

Most importantly, at final performance all 19 trainer models led to the worst possible return with $\gamma = 1$. With $\gamma = 0.99$, 18 models led to minimal return. We visually examined agents learning at $\gamma = 1$ from five of the trainer models; each

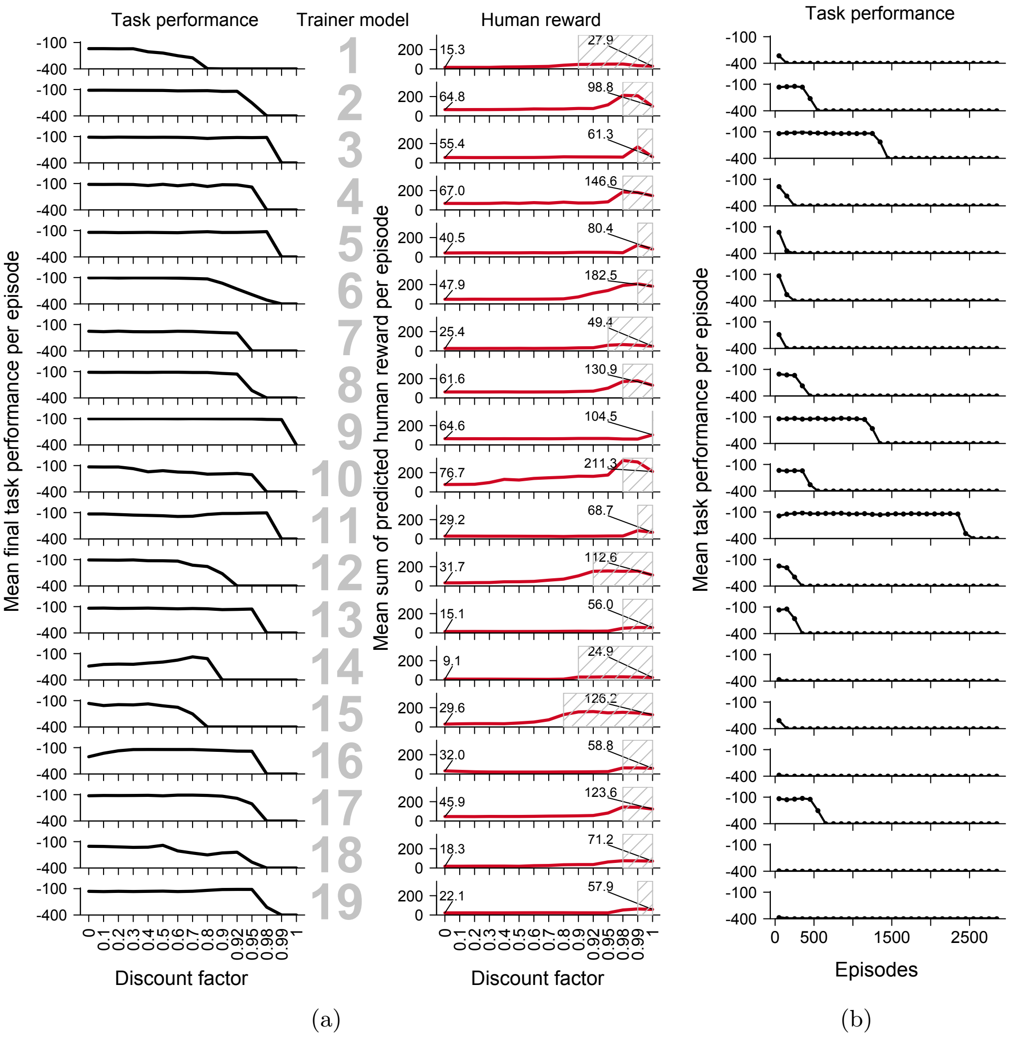


Figure 6.2: (a) Non-aggregate version of the results in Figure 6.1, showing learned task performance over 500 episodes after 3500 episodes of learning (left) and mean total reward per episode over these final 500 episodes (right) for each of the 19 trainer models. Gray shading in the "Human reward" column indicates the inclusive range of γ values at which the agent's task performance is minimal (-400 per episode). (b) Learning curves at $\gamma = 1$, showing mean task performance over 100 episode intervals for each trainer model.

agent exhibited a circuitous behavior, indicating the positive circuits problem is likely responsible for minimal task performance. Indeed, the mean sum of predicted human reward per episode increases as performance decreases, as can be seen in the plots of the final task performance with each trainer model (Figure 6.2). For all 19 trainer models, the mean reward accrued per episode is higher at discount factor of 1 than 0. Further, for almost every trainer, at every γ value that leads to worst-possible task performance (i.e., values shaded gray in the "Human reward" column of Figure 6.2), the corresponding mean total reward per episode is higher than at all γ values that lead to better performance. The three exceptions (trainer models 2, 3, and 6) break this general observation by small margins, 15% or less.

Two general patterns emerge. I have noted the first: task performance decreases as the discount factor increases. Secondly, agent algorithms also accrue higher amounts of predicted human reward as the discount factor increases. In other words, best task performance is not aligned with behavior that accrues the most predicted human reward.

Figure 6.2(b) shows learning curves at 100-episode intervals for a single run at $\gamma = 1$ for each trainer model. Good initial performance lasts for a varying amount of time but then degrades to worst-possible performance quickly. In the plots, this degradation occurs during the intervals with intermediate performance.

In general, the choice of RL algorithm will impact performance, so one might ask whether the algorithm used here is actually learning an MDP-optimal policy for its corresponding human reward model and discount factor. At $\gamma = 0$ and $\gamma = 1$, the answer appears to be "yes." At $\gamma = 0$, the agent optimizes return at tree search depths greater than 0. When the search depth is zero, it uses the learned value for Q(s,a), which is roughly equivalent to $\hat{R}_H(s, a)$ after many learning samples at or near (s,a). At $\gamma = 1$, if the RL algorithm learns an infinitely repeatable sequence of actions with positive net reward, then the disastrous policy that repeats that sequence is necessarily within the set of MDP-optimal policies. As mentioned above, we visually checked the behavior of five models' corresponding algorithms while they exhibited the worst possible performance, and each agent repeatedly visited roughly the same circuit of states until the episode limit was reached. During this circuitous behavior, the maximum Q values at all observed states were positive. Therefore, the results for $\gamma = 0$ and $\gamma = 1$ can be considered correct — independent of the RL algorithm used — with confidence. However, for $0 < \gamma < 1$, another RL algorithm might learn a policy with a higher mean total reward per episode than in these results.

There is one important caveat to the conclusions we draw from this γ-independent model analysis. Training occurred with TAMER algorithms, effectively at $\gamma = 0$. We strongly suspect that trainers adjust to the algorithm with which they interact; if the agent is maximizing immediate reward, a trainer will likely give more reward for immediately previous behavior. Only a γ-dependent model analysis—as we perform in the following experiment—will address whether this caveat of trainer adjustment has affected our conclusions.

6.4.2 Setting γ before training human reward models

In the analysis described in this section, as in the γ-independent model analysis in the previous section, the human reward model $\hat{R}_H$ is learned by TAMER and provides predictions that are interpreted as reward by an RL algorithm. But unlike the previous analysis, $\hat{R}_H$ is learned while performing reinforcement learning, and the RL algorithm, not TAMER, selects actions while learning $\hat{R}_H$. Thus the human trainer will be adapting to the same algorithm, with the same γ, that is being tested.

Because the agent in this experiment learns from a frequently changing reward function, behaving optimally with respect to the current reward function is difficult. Our choice of task and RL algorithm creates approximately MDP-optimal

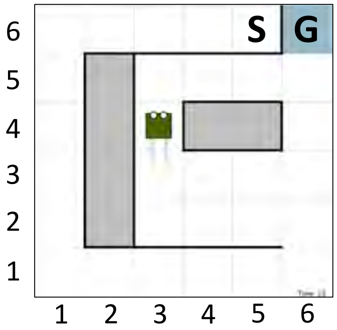


Figure 6.3: A screenshot of the grid world task used in the experiments in Sections 6.4.2 and 6.5. To display the agent's actions and state transitions, the simulated robot's eyes point in the direction of the last action and wheel tracks connect the agent's last occupied cell to its current location. The start and goal cells are labeled 'S' and 'G' respectively.

behavior with small lag in responding to changes to the reward function, a lag of a few time steps or less.

The task is a grid world with 30 states, shown in Figure 6.3. At each step, the agent acts by moving up, down, left, or right, and it cannot pass through walls. Task performance is measured as the time steps taken to reach the goal. The agent always starts a learning episode in the state labeled "**S**" in Figure 6.3. The shortest path from the start state requires 19 actions. Each time step lasts approximately 800 ms.

The reinforcement learning algorithm is value iteration [90] with greedy action selection, except that instead of iterating until state values converge, one update sweep over all of the states occurs each 20 ms, creating 40 sweeps per step. At each step, the agent greedily chooses the action that maximizes predicted return for the current state, as calculated by a one-step lookahead for each action, using the predicted human reward and discounted value of the next state.

The TAMER module, which learns the human reward model $\hat{R}_H$, represents $\hat{R}_H$ as a linear model of Gaussian RBFs. One RBF is centered on each cell of the grid world, effectively creating a pseudo-tabular representation that generalizes slightly between nearby cells.[5]

The experiments were conducted through subjects' web browsers via Amazon Mechanical Turk. Subjects were randomly given an algorithm using one of five different discount factors: 0, 0.7, 0.9, 0.99, and 1. For these five conditions, the respective number of subjects was 10, 8, 10, 7, and 7.[6] Subjects were prepared with video instructions and a period of controlling the agent followed by a practice training session. The actual training session stopped after the agent reached the goal 5 times or after 300 steps, whichever came first.

Figure 6.4 shows the success rate of trained agents by condition, dividing them among those that never reach the goal, reach the goal 1–4 times, and reach the goal the maximum 5 times. These results exhibit a clear pattern of task performance worsening as the discount factor increases. This pattern is supported by significance testing. Fisher's Tests compared outcomes of reaching the goal all 5 times or not by condition: between $\gamma = 0$ and $\gamma = 1$, $p = 0.0006$ (extremely significant); between $\gamma = 0$ and $\gamma = 0.9$, $p = 0.0325$ (significant); and between $\gamma = 0$ and $\gamma = 0.7$, $p = 0.4444$ (not significant).

To evaluate reward positivity, the basis for the positive circuits problem (Section 6.3), we examine the ratio of cumulative positive reward to cumulative negative reward given by successful trainers in each condition, shown in Figure 6.5. Success appears highly related to this ratio; in Figure 6.5, we are able to draw a

[5] Each RBF has a width $\sigma^2 = 0.05$, where 1 is the distance to the nearest adjacent RBF center, and the linear model has an additional bias feature of constant value 0.1. $\hat{R}_H$ is updated with new feedback by incremental gradient descent with a step size of 0.2. In accordance with the most recent version of TAMER (Section 3.3), we used aggregate reward for credit assignment with a probability distribution over feedback delay of Uniform(-0.4 seconds, -0.15 seconds) (with negative values because the algorithm looks backwards in time from the feedback signal to potentially targeted events), and updates occurred at every step regardless of whether reward was provided.

[6] In the two Mechanical Turk experiments (here and in Section 6.5), variation in subject numbers comes from a few user errors (usually not typing in their provided experimental condition correctly), errors in logging training data, and the removal of subjects for insufficient feedback. In the first and second experiments, 3 and 2 subjects were respectively removed for insufficient feedback. Of these 5 subjects, only 1 gave any feedback during the entire training session. This trainer had a feedback-instances-to-time-steps ratio of 0.01.; subjects who were retained had ratios above 0.1.

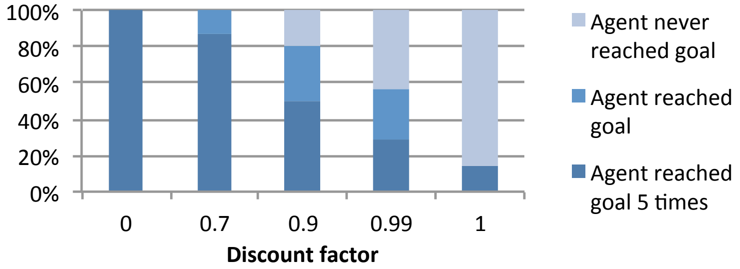


Figure 6.4: Success rates for the episodic grid-world experiment by discount factor.

dividing line at each condition between all agents that never reach the goal and all other, more successful agents. Additionally, the ratio of this division between success and failure monotonically decreases as the discount factor increases, which supports our conjecture that the positivity of human reward becomes more problematic as the discount factor increases (Section 6.3). Without recognition of the positive circuits problem (Section 6.3), this pattern of lower-performing agents getting *more* reward would be quite counter-intuitive. Further, negative Spearman correlations between discount factor and these ratios are extremely significant both for all trainers and for only trainers whose agents reached the goal once or more ($p <= 0.0005$), but the correlation when considering only goal-reaching trainers is stronger (correlation coefficient $\rho = -0.7594$, compared to $\rho = -0.543$ for all trainers). We conjecture that γ affects ratios by both filtering out trainers that give too much positive reward in conditions of higher γs and by pressuring trainers to adjust their ratio in response to the agent. In surveys given after training, at least one trainer, from the $\gamma = 0.9$ group, spoke of his attempts to adapt to the agent: "When [the reward] key is stroked there is not much response in the robot. Only [the punishment] key stroke worked."

Reward is predominately positive (a ratio greater than 1) for 66.7% of trainers in this experiment, which supports the conjecture that human reward generally has a positive bias. At $\gamma = 0$, all training is predominately positive, fitting results

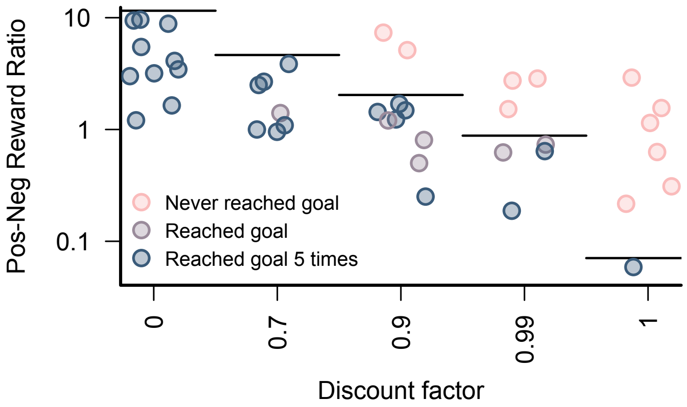


Figure 6.5: Ratio of cumulative positive reward to cumulative negative reward given by each trainer, divided by discount factor condition and task performance. Jitter has been added along the x-axis for readability. For each condition, a horizontal line was placed above the mark for the highest ratio at which a subject trained the agent to reach the goal at least once.

from TAMER experiments (Section 3.4.3). However, across all γs we find a higher incidence of predominately negative training than did these TAMER experiments and past work [107], possibly because these ratios are the first reported for learning from human reward with high γ values. After training, *there was at least one behavioral circuit with net-positive reward in 35 of the 42 MDPs created from trainers' reward models*. In other words, 83.3% of the trained agents would exhibit the positive circuits problem if learning with $\gamma = 1$. Half of the predominately negative trainers created positive circuits. Those without positive circuits all had positive-to-negative reward ratios below 0.63 and generally were from higher γ conditions: one from 0.7 and two each from 0.9, 0.99, and 1.

Taken together, the two experiments in this section provide strong evidence that agents should act myopically (i.e., use small discount factors) when learning goal-based, episodic tasks from human reward.

6.5 Empirical results - discounting for tasks modified to appear continuing

In this section, we explore an alternative approach to addressing the positive circuits problem, our conjecture that a positive reward bias among human trainers combined with high discount factors can lead to infinite behavioral circuits (i.e. loops)—and thus minimal task performance. Importantly, this issue is specific to episodic tasks, where an agent reaching the goal is effectively penalized by no longer being able to receive reward. Another strategy is to remove this penalty: convert the episodic task to a continuing one. Unlike for episodic MDPs, the optimal policy of a continuing MDP is unaffected by adding a constant value to all reward; thus, the positivity of human reward should not present the same problem in continuing tasks.

6.5.1 Continuing task experiment

To convert task dynamics from episodic to continuing, we add to the agent's experience transitions that are created by replacing the absorbing state at the end of each episode with the start state of the next episode. Consequently, reward received in one episode can be attributed to state-action pairs in the previous episode (and farther in the past). More precisely, the original task's transition probability from any state-action pair s, a to a state s' is increased by the probability that s, a would have reached an absorbing state times $D(s')$, the probability of s' being sampled as a starting state. The probability of transitioning to an absorbing state becomes 0.

We repeated the experiment in Section 6.4.2 almost exactly, only changing the task dynamics as described above. Additionally, subjects trained for 10 episodes or 450 time steps, whichever came first. For consistency with the previous experiment, we analyze data from the first 5 episodes that occur before the 301st time step. For γs of 0, 0.7, 0.9, 0.99, and 1, there were respectively 10, 8, 9, 10, and

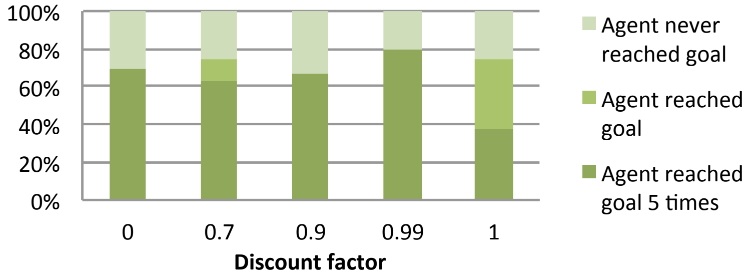


Figure 6.6: Success rates for the continuing grid-world experiment by discount factor. Note that all plots related to the episodic grid-world experiment are blue and those related to the continuing grid-world experiment are green.

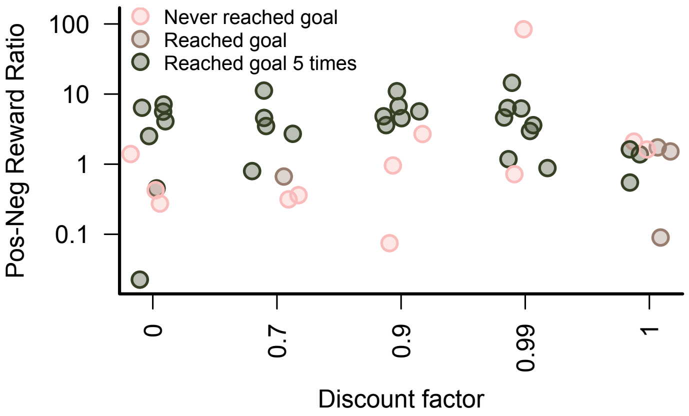


Figure 6.7: For the continuing grid-world experiment, ratio of cumulative positive reward to cumulative negative reward given by each trainer (with x-axis jitter).

8 subjects. Figures 6.6 and 6.7 show results (presented analogously to Figures 6.4 and 6.5).

In comparison to Section 6.4.2, the conversion to a continuing task creates markedly different results. The task success rate at $\gamma = 1$ is lower than at other conditions, which we expect given that this discounting is generally avoided for continuing tasks to make rewards in the finite future meaningful. The other discount factors create similar task performance, with $\gamma = 0.99$ achieving the highest mean success rate. Fisher's Tests, again comparing outcomes of reaching the goal all 5 times or not by condition, find that no condition is significantly different than another ($p > 0.14$ for all pairwise comparisons). Note that the conversion to continuing does not affect a $\gamma = 0$ agent. The difference in success rate at $\gamma = 0$ in the two grid world experiments is likely because of either randomness—their difference is also insignificant by a Fisher's Test ($p = 0.2105$)—or differing times of day at which we ran the experiments, possibly sampling from a lower-performing population for this experiment.

Patterns exhibited by the ratios of cumulative positive reward to cumulative negative reward among trainers also differ from the episodic experiment. There is no correlation between the ratios of fully successful trainers and discount factor (Spearman coefficient $\rho = -0.0881$, $p = 0.6493$). Further, among the $\gamma < 1$ discounting conditions, the relationship between reward positivity and task performance is closer to the intuitive expectation that high-performance agents receive more positively-biased reward.

Converting the task to continuing does indeed appear to remove the adverse impact of reward positivity at high discount factors, overcoming the positive circuits problem. However, based only on the roughly equivalent task performance for all $\gamma < 1$ conditions, the choice of which discounting to use is unclear. In the next subsection, we investigate whether learning is more robust to changes to the environment or more general at certain γ values.

6.5.2 Benefits of higher γ values

In Section 6.2, I point out that at γs near 1, reward can communicate a desired policy, the goals of the task, or a mix of the two. Using the full training data from this experiment (up to 10 episodes or 450 time steps), we now investigate whether the trained agents do learn more than a policy. Since $\gamma = 1$ is generally inappropriate for continuing tasks, we expect $\gamma = 0.99$ to yield the best results. We

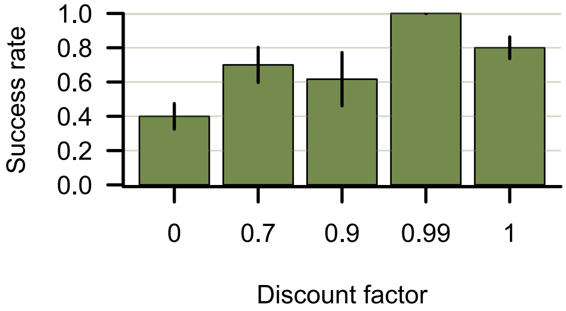


Figure 6.8: Mean rate of successfully trained agents reaching the goal in less than 100 time steps from the 10 states off of the optimal path. Standard error bars are calculated using a single agent's success rate as one sample.

restrict our analysis to those agents successfully trained to get to the goal 5 times in less than 300 steps. Thus, we effectively ask, given that an agent learns a good (and usually optimal) policy from human reward, what else does the agent learn?

We first test the learned policy from 10 states that are not along the optimal path from the start state (i.e., the states not on the border). These states may have never been experienced by the agent during training, in which case $\hat{R}_H$ is built without any samples from the state. Simple policy generalization from nearby, optimal-path states would help in only some of these 10 states, so the ability of the agent to get to the goal reflects whether the agent has learned some information about the task goals. Agents that had learned their policies at higher γs were more often able to get to the goal in less than 100 time steps (Figure 6.8). Every successfully trained agent in the $\gamma = 0.99$ condition reached the goal from every tested state. Note though that different discount factors might lead to different levels of experience in these tested states, providing a confounding factor.

In the second test, an obstacle is placed in the state two cells below the goal (column 6, row 4 in Figure 6.9), blocking the optimal path, and we then determine whether the agent can still reach the goal in less than 100 time steps. Thus, we test the effects of changing the task-optimal policy but keeping constant the task goal:

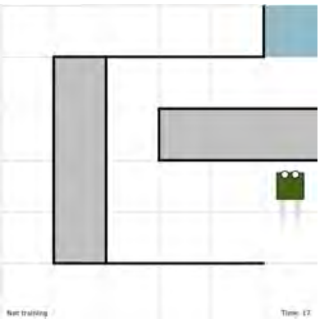


Figure 6.9: A screenshot of the grid world task with an obstacle added for the second analysis of Section 6.5.2.

get to the goal state as quickly as possible. At state-action pairs that previously transitioned into the newly blocked state, the agent's reward function is modified to output 0 to reflect the agent's lack of knowledge about how the trainer would reward these transitions. In the $\gamma = 0.99$ condition, 4 of 8 agents reached the goal. No agents with other γs did. We additionally tested agents using human models learned after 1, 2, ... episodes of training, instead of only after training. In these expanded tests, 8 of the 8 successful $\gamma = 0.99$ agents could reach the goal at some point between episodes of training, as could 1 of the 3 successful $\gamma = 1$ agents; no other agents reached the goal.

These analyses support the conjecture that agents taught with higher discount factors may learn about the task goals itself, making the agents generally more robust to changes in the environment and more able to act appropriately in previously unexperienced states. That agents may learn task goals raises the tantalizing prospect that, under the right circumstances, an agent receiving reward from a human trainer could learn a policy that is far superior to that envisioned by the trainer.

6.6 Disadvantages of higher γ values

In domains that are more complex than this grid-world task we have used for experiments, we may be unable to use value iteration with iterating sweeps over the entire state; even ignoring the possibility of continuous states or actions, some tasks simply contain too many states to quickly and repeatedly perform temporal difference updates on all states. In anticipation of scaling the high-gamma, continuing-task approach found successful in Section 6.5.1, we implemented a version of the Monte Carlo tree search algorithm Upper Confidence Trees (UCT) which has been successful in tasks with especially large state spaces [53], originally in games like Go [30] but also in more general reinforcement learning tasks [33].

Because these are merely initial results, I do not go into the specifics of the UCT algorithm. What is important to know, however, is that the agent “plans” by repeatedly considering different possible 10-step trajectories from its current state. During a single time step, numerous unique trajectories are generated, and each transition within a trajectory generates a temporal difference update to the agent’s action-value function, as if the agent had actually experienced the transition. (In fact, real transitions are discarded and not used for updates, since they will have already been “experienced” many times by the agent during planning.) Therefore, the number of updates for each state are not equivalent, as they are between sweep iterations in our value iteration implementation. Instead, state transitions that can quickly be reached from the current state receive many more temporal difference updates than transitions from less “local” states. For complex tasks in general, this bias towards local updating appears desirable, since an effective learning agent will likely visit regions of the state space that are worth understanding more often than areas that can be ignored during learning. Additionally, this local updating bias occurs in a large fraction of common RL algorithms (e.g., SARSA(λ), Q-learning, and Monte Carlo tree search algorithms).

In training sessions performed by the author, this algorithm is *much* less effective on the same continuing grid-world task than value iteration. At $\gamma = 0$, there is no difference, as one would expect, since the agent finds the MDP-optimal policy easily. But we are interested in high values of γ. At $\gamma = 0.9$, the author could train the agent to reach the goal after much frustrating back-tracking, which is not a characteristic of the value iteration approach. Specifically, any time the agent starts to make new ground on the optimal path, it would quickly reverse directions within 1–3 steps, allowing it to revisit the transitions that it recently received positive reward on. At $\gamma = 0.99$, the agent could only be trained to reach the goal with highly negative reward—about 3 negative button presses per incorrect action and one positive press per correct action—which, as this chapter reveals, cannot be expected of human trainers in general. Even with this uncommonly negative reward, the agent was quite slow to train; for $\gamma = 0.99$ with value iteration (and likewise the continuing version of this task), the author can consistently train the agent to follow the optimal policy after less than 10 incorrect actions, but the UCT agent makes many more incorrect actions than correct ones before reaching the goal its first time.

From initial analysis of these results, we believe we are encountering an additional problem caused by the overwhelming positivity of human reward—a problem that is specific to RL algorithms with a local-updating bias and is not solved by making the task continuing. The problem appears to be that the UCT agent—with an action-value function initialized to 0 as in the other experiments of this chapter—encounters enough positive reward that those transitions that receive more updates than others appear to have a higher value and are consequently experienced *even more* in future learning, to the extent that the agent never learns the true values of some states along its task-optimal and MDP-optimal path. In other words, the agent simply learns to keep doing what it has already done, almost regardless of what those past actions are.

One apparent solution, optimistically initializing the action-value function, is not an option in complex domains for two reasons. First, optimism leads to thorough exploration of the state-action space. Exhaustive exploration during training would frustrate, exhaust, and confuse the trainer, since such exploration would include much behavior that goes against the trainer's feedback, making the agent appear unresponsive and unable to learn for a considerable period. Thorough exploration would also sacrifice the fast learning that is one of the chief appeals of interactive shaping. Planning-only exploration from optimistic initialization, where the agent's actual actions are greedy, might be possible, but it would be greatly complicated by the second reason that optimistic initialization is problematic. This reason is that the reward function is constantly changing during training. If the agent reinitializes its action-value function optimistically each time step, it forfeits all knowledge gained during previous time steps about action values under similar reward function, knowledge that should be critical to learning quickly to perform well with the new reward function. On the other hand, if the agent only initializes at the beginning of training, then it will not explore with new reward functions, largely removing the impact of optimistic initialization.

In summary, we expect that agents with biases towards local updates—agents often well-suited for complex tasks—will ineffectively learn from human reward even in continuing tasks; the problems of reward positivity extend beyond episodic tasks. We are, however, optimistic that these problems can also be solved with further investigation.

6.7 Summary

Given a reward function—a model of human reward in this chapter—the choice of γ and whether the task is experienced as episodic or continuing determine the MDP-optimal policies. This chapter investigates which settings create the MDP-optimal policies that perform best on a task performance metric τ that guides the trainer's feedback. The insights of this chapter arise from the positive circuits problem, caused by a bias towards positive rewards among human trainers (Section 6.3). The empirical results described in Section 6.4 indicate that MDP-optimal policies defined by $\gamma = 0$ discounting translate to the best task performance for episodic tasks. However, our experiment in Section 6.5 indicates that converting an episodic task to a continuing one removes the adverse impact of reward positivity; task performance was roughly equivalent across all conditions where $\gamma < 1$. Additionally, post-hoc testing of these trained continuing-task agents (Section 6.5.2) suggests that high, non-one discount factors, like $\gamma = 0.99$ for our grid world task, yield better task performance on more complex tasks, where the trainer does not know the optimal policy, and on tasks with changing dynamics. However, in Section 6.6 we identify an additional challenge to using γ values near one that can occur even in continuing tasks and will need to be addressed to learn effectively in complex tasks.

From these experiments, we submit two practical hypotheses for how to best learn from human reward: (1) *human reward should always be given in continuing settings (or other settings where reaching a goal does not penalize the agent, such as learning to maximize average reward)*, and (2) *given such continuing settings, discount factors less than but near 1 should be used.*

Unlike research that shows what an algorithm can do, the contributions of this chapter rest on what an algorithm *cannot* do and a manipulation of the task that removes the limitations of otherwise incapable algorithms. To this end, the simplicity of the grid world task is critical: if an agent cannot learn an effective policy in a 30-state grid world, it will likely fail in more complex tasks.

In addition to lowering the discount factor or converting the task to continuing, there are other candidate strategies to address the positive circuits problem.

For instance, mapping all human reward to negative values would communicate that the task is goal-based; however, this mapping is not an option because we seek algorithms that are agnostic to whether the task is goal-based. For the same reason, we do not consider manipulations aimed at communicating the goal-based nature of the task, such as adding large reward when reaching any absorbing state. On the other hand, acting based on expected returns of average reward, an approach often used in continuing tasks, does appear to address the positive loops problem without biasing learning towards assumptions that the task is goal-based.

In summary, this chapter contains five main contributions:

- identifying the trend in past work of using high discounting rates;
- linking human reward positivity to positive circuits, empirically establishing positive loops' prevalence, and giving resultant algorithmic guidance (i.e., either use low discount factors or only learn discounting tasks);
- relating γ, human reward positivity, episodicity, and task performance in goal-based tasks;
- adding structure to the body of past work on learning from human reward by framing and analyzing the question of how human reward should be discounted;
- and providing the first instance of successfully learning from human reward with low discounting ($\gamma = 0.99$ with 0.8 second time steps).

Here concludes the core technical chapters of this thesis. Revisiting the Interactive Shaping Problem that Chapter 3 addresses, this chapter validates TAMER's myopic approach for episodic domains and provides direction towards even more powerful algorithms when the task is or can be interpreted as continuing. In the following chapter—which concludes this thesis—I give differing prescriptions for how to learn from human reward in practice, given current knowledge of the topic, and in research.

Chapter 7

Conclusion

In this thesis, I consider deeply the question of how agents should learn from human-generated reward. I formalize this problem as the Interactive Shaping Problem in Chapter 2. A candidate solution for interactive shaping must contain two high-level components: (1) a mapping from agent behavior and human reward to some objective function, and (2) an algorithm that optimizes or otherwise performs well on the chosen objective. Any solution to the problem of interactive shaping can be evaluated by the performance of a trained agent, as measured by a task performance metric that is understood by the trainer.

The TAMER framework, presented in Chapter 3, is one solution. A TAMER agent models the human trainer's reward and generally acts to maximize the amount of reward directly attributable to that action, without consideration of the action's effect on future state. In that chapter, I present results from user studies that verify the applicability of the TAMER framework to a range of tasks and demonstrate that interactive shaping through TAMER—during early learning stages—can dramatically outperform agents that learn from hard-coded feedback functions (e.g., a reward function that is part of a Markov Decision Process).

Chapter 4 builds upon the TAMER framework by combining its algorithms for learning from human reward with a reinforcement learning algorithm that learns from MDP reward, learning from both feedback signals. With both feedback signals available, we examined eight methods for combining a TAMER-learned model of human reward with the RL algorithm. Two methods consistently outperform learning from only one of the two signal types: action biasing and control sharing. Both of these methods only change the RL agent's action selection, effectively using the human model $\hat{R}_H$ to provide explorative experience for the agent.

In Chapter 5, we again employ the TAMER framework, this time investigating the effects of two manipulations on trainers in carefully controlled, randomized experiments with a relatively large number of subjects. The first manipulation focuses on the instructions given to the trainers, specifically comparing trainers who know they are teaching an agent against those who are told that they are merely critiquing a recording (though they are in fact also teaching). The results of this experiment indicate that the manipulation had little effect on the amount of feedback given or the agent's learned performance. In the main experimental condition of the second experiment, the agent chooses actions that it believes to be of low quality whenever the trainer's recent feedback frequency becomes low. This reactive "misbehavior" resulted in more overall feedback but worse performance by the agents. These experiments are discussed as early examples of using learning agents in the place of humans in experiments concerning human behavior.

Chapter 6, containing the final core technical contributions, reexamines the pure interactive shaping problem from the context of TAMER as well as other work on the problem. In this chapter, the mapping from agent behavior and human reward to an objective function is explored. We generalize from and question the myopic assumption of the TAMER framework; the model $\hat{R}_H$ is learned, but we test different levels of discounting predicted future rewards. We identify the *positive circuits problem* of episodic, goal-based tasks that arises from the positivity of human reward, giving an intuitive explanation for the existence and disastrous effects of positive circuits as well as demonstrating their prevalence in a user study. In this study, the myopic approach of TAMER (i.e. $\gamma = 0$) leads to higher task performance than does optimizing behavior for higher discount factors that are more typical in reinforcement learning ($\gamma \geq 0.9$). We also demonstrate, in another experiment, that converting the episodic task to one that appears continuing partially solves the positive circuits problem by removing the penalty that an agent incurs by reaching the goal—an inability to gather further reward in a reward-positive world. With a continuing task formulation and an RL algorithm that quickly learns approximately optimal behavior, all discounting levels result in similarly successful training. Finally, we show that less myopic discounting has powerful advantages—the agent can use task information in the human reward signal to adapt to environmental changes and act appropriately from states in which it has limited experience—and challenging disadvantages—even in continuing tasks, the positivity of human reward biases agents that learn mostly from real or simulated experience towards repeating past actions.

Current best practices for learning from human reward In introducing TAMER in Chapter 3, I argue that human reward *can* be learned effectively from with a $\gamma = 0$ assumption. In Chapter 6, we examine whether human reward *should* be learned according to this discounting assumption. We find that within the assumptions made previously—especially that an episodic task should be learned from as an episodic task—the answer is "yes" for at least a large class of tasks. But by replacing the assumption about episodicity with an assumption that episodic tasks should be converted to continuing ones, we find that the identified issues with higher γs disappear in a simple grid world task in which the MDP-optimal policy can be quickly found. Consequently, the choice of best discounting becomes less clear.

At $\gamma = 0$, a higher burden is placed on the trainer, who must teach a policy rather than potentially simpler task goals, requiring more work. On the other hand, the algorithm for finding the MDP-optimal policy at $\gamma = 0$ is trivial. At γs near 1, the burden is more heavily placed on the algorithm. The trainer can teach task goals relatively quickly by giving reward that is planned for farsightedly. But initial results reported in Section 6.6 suggests that the positivity of human reward creates a challenge for many reinforcement learning algorithms even in continuing tasks.

Until further investigation, I believe that the use of higher discount factors—where TAMER is a module that learns a human reward function within a larger RL algorithm rather than TAMER forming the top-level agent algorithm—is a more promising direction of future algorithmic research. But to get high-quality results on complex tasks with small action spaces, TAMER still appears to be the best option for now.

7.1 Directions for future work

A goal of this thesis is to establish interactive shaping as a teaching technique of considerable promise and, consequently, as a worthy focus of future research. An exciting aspect of this thesis is that it indeed opens numerous avenues of inquiry. I briefly describe some of these research directions below, starting with research that strictly addresses the problem of interactive shaping.

- **Trainer preparation** - Subjects who train agents in our experiments were prepared through some form of instruction—written directions, live verbal directions, and/or an online video—and some practice—controlling the agent and/or training it—before conducting the training session(s) that are counted in our analysis. The specifics of preparation surely have some effects and deserve careful analysis. Further, the amount of preparation appears to have a significant impact [51]; the best algorithms for an expert, professional trainer will likely differ from those for casual or novice trainers.

- **Transparency** In the displays shown to trainers in our implementations of interactive shaping in simulation, the most recent state and current action are shown. However, an agent may contain further information that would be worthwhile to share with the trainer. Such information includes confidence about reward predictions and intended behavior. Given transparency of intention, user feedback might be applied to intended action as well as past action, an approach we have begun to explore in preliminary work [49]. By allowing the punishment of intention, the agent could learn from "mistakes" without necessarily making them.

- **Interfaces for giving reward** - In our experiments, we always use push-buttons to deliver reward. We do so for two reasons. One, such an interface is easy to create and deploy. Two, the simplicity of a push-button interface makes it intuitively seem to be a decent proxy for other interfaces, such as vocal feedback and facial expression. However, other interfaces may seem more natural and should be tested. Others have used mouse gestures [107, 88] or speech converted to text [101]. Interestingly, a mapping of prosodic features such as tone and volume of speech to human reward values has been learned effectively in past work [45]. However, to my knowledge no research has explored the comparative utility of different interfaces.

- **Mappings from user input to reward values** - Trainer feedback is given by push buttons and mapped to $+1$ and -1 in our experiments. This mapping fits previous work [107, 101, 76, 88] and is intuitively satisfying. However, other mappings should be examined, specifically to $+1$ and $-c$, for various values of c. When $c > 1$, the positive circuits problem (see Chapter 6) may be alleviated or even removed, though making negative rewards more severe might introduce other problems, such as complete avoidance of areas of state-action space that are accidentally punished just once.

- **Personalization of the learning algorithm** - Interactive shaping allows trainers to personalize the behavior of agents to fit their desires. Additionally, the algorithms for interactive shaping might themselves be personalized to each user. For example, people will differ in their distribution of feedback delays, which is critical for TAMER's credit assignment (see Section 3.3). Calibration—either before or during training—might yield improvement in agent trainability.

- **Non-Markovian models of human reward** - TAMER's credit assignment module, described in Section 3.3, can be seen as a heuristically guided system for attributing causality of reward signals to specific events, namely state-action pairs. However, this approach assumes that the expectation of human reward can be represented such that state-action pairs are independent of each other. In practice, this appears to be a limiting assumption. For example, consider an agent that is quickly oscillating between left and right movements, never leaving the same two states. If the trainer wants the agent to continuously move right, he or she might give strong negative reward to the oscillatory behavior—composed of iterations between correct and incorrect actions—despite the occurrence of the correct behavior in part of the oscillations. The trainer could be more accurately modeled by a representation that can differentiate between a rightward action that is part of consistent rightward movement and a rightward action that is part of such oscillation. Conversely, sometimes such oscillations are desirable, as in the balancing cart pole task (Section 4.3.5). Creating a k-order Markov model of human reward, adding specific features of history (e.g., the previous action), or learning predictive state representations [61] are possible methods for modeling trainers' responses to events composed of multiple actions or state transitions. When $\gamma > 0$ (see Chapter 6), such an approach will make planning more difficult, but planning with non-Markovian reward and transition functions has already received some focus [10], which we might draw on.

- **Modeling reward in large state spaces through automatic dimensionality reduction** - The state-action features that are thus far input to various $\hat{R}_H$ representations are hand-chosen. However, a large body of research exists on reducing dimensionality for regression algorithms. Any interactive shaping should seek to both learn desired behavior quickly and be flexible enough to learn a wide range of behaviors. Any one desired behavior might only rely on a few features; having many possible features grants the flexibility to learn different behaviors, each requiring only a few features, but some form of dimensionality reduction may be necessary to maintain fast learning. I suspect that supervised methods—such as L1 and L2 regularization and regression trees that branch based on entropy reduction—will be more effective than unsupervised ones, since supervised methods consider the actual reward being provided and thus can select or create features that enable prediction of training labels. Also, note that this research extension could be addressed in concert with creating non-Markovian models of human reward.

- **Scaling reinforcement learning algorithms for low discounting of predicted future reward** - As I discuss in Section 6.6, challenges remain in applying RL algorithms to complex tasks with discount factors near 1. This direction of research is possibly the most critical of those listed here; if robust and effective algorithms can be identified or developed, I believe the applicability and benefit of interactive shaping will increase greatly.

- **Implement in application domains** As of yet, interactive shaping has been applied to simulated and robotic testbed domains. Interactive shaping is meant for real-world tasks and it should be eventually applied to aid their learning. One potential domain is neural prosthetics, in which signals from a human's motor neurons are used to control the motors of a prosthetic device (e.g., a prosthetic limb) [76]. These prosthetics often need calibration and personalization, and the use of interactive shaping could empower users to adjust their prosthetics to their own needs. Another potential domain is brain-computer interfaces (BCIs). For instance, researchers are investigating how to use data from electroencephalograms (EEGs) on quadriplegics to control assistive devices, such as robotic arms. Given the low bandwidth of information flow with current EEG interfaces, interactive shaping is a promising method for enabling users to train assistive devices to perform certain behaviors, such as grabbing the object that the person is visually focusing on (assuming an eye tracker is also in use). Lastly, human-robot collaboration is a highly active area of research. For many collaborative tasks, the human teammate will have user-specific demands on the robots' behavior. Natural teaching methods like interactive shaping—without requiring programming skills—may be powerful avenues for providing such flexible and easy designation of behavior.

Each item in the following list of potential research topics involves aspects of interactive shaping—learning from intentionally delivered human reward signals—but differs from the pure Interactive Shaping Problem.

- **Unintended rewards** - Human rewards given without the intention to teach or otherwise affect behavior—possibly derived from smiles, attention, tone of voice, or other social cues—are more abundantly broadcast and can be observed without adding any cognitive load to the human. And like intentional rewards, unintended signals contain information about how an agent's behavior is affecting the human source. Therefore, learning behavior or at least fine-tuning it from such signals is a crucial and immensely promising direction of research. However, such rewards are untargeted; someone might be smiling for any number of reasons that have nothing to do with the agent. Consequently, interpretation and attribution of these social cues will be especially challenging. It also remains to be seen how much the insight gained about learning from intentional reward is applicable to learning from unintentional reward.

- **Revisiting TAMER+RL** - As we improve our understanding of how to learn effectively from human reward, new TAMER+RL techniques will become apparent. Specifically, the examination of discounting in Chapter 6 provides insight on why reward shaping is less effective than action biasing. Following the formal unification of the two methods in Section 4.3.4, reward shaping is equivalent to learning and combining two Q functions: one from MDP reward and one from human reward with the MDP's discount factor, which in Chapter 4 is $\gamma = 1$ in two different episodic tasks. Chapter 6 shows such discounting to be catastrophic when learning to perform goal-based, episodic tasks without an additional MDP reward signal. Thus, a new method in which one Q function is learnt regularly with MDP reward and another is learnt from human reward at a different discount factor—where the agent's experience is manipulated to appear as if it comes from a continuing task—may outperform the best of the current TAMER+RL combination techniques.

- **Integrate interactive shaping with other natural teaching methods** - Except for TAMER+RL in Chapter 4, I have kept focus on learning only from human reward as a research philosophy, hoping that such a focused and reductive approach would lead to greater insight and progress. However, as I express repeatedly throughout this dissertation, interactive shaping is not meant to be a monolithic method of teaching. In addition to combining human and MDP reward as Chapter 4, the combination of demonstration and human reward seems particularly powerful. For a such a learning algorithm, an appealing teaching strategy is to demonstrate first to reach a decent but imperfect behavior and then use human reward to tune the behavior further. Indeed, as I describe in Section 2.5, Koachar et al. found that people tended to give demonstration followed by feedback in a Wizard of Oz study [42].

- **Hidden state** - This thesis has not touched on the topic of hidden state—for instance, as in a partially observable Markov decision process (POMDP) formulation—a characteristic of many real-world problems. This topic would likely overlap with the research direction "Non-Markovian models of human reward" above.

7.2 Long-term vision

The path of the fields of artificial intelligence and robotics will be influenced by what types of information we can harness to learn desirable behavior. If robots and other agents are left to learn only through trial and error with programmatically predefined objectives, they may remain incompetent; but, in my view, the real danger is that they become competent and productive but remain detached from human needs.

Robots and other agents are poised to dramatically alter demand for human employment, both creating and removing jobs. If the benefits of this new technology are to be widely distributed—beyond only the technologically elite and those who do business with them—we researchers must develop algorithms that effectively place humans in the loop.

Whereas artificial intelligence is often dominated by visions of autonomy, this thesis reaches for a human-centered AI, where humans do not passively receive the benefits of AI but rather actively understand and exert control over the behavior of the agents. The development of interactive shaping—through the TAMER framework and otherwise—nudges the field of AI in this direction, empowering human users to teach behaviors and improving their understanding of the agents through the interactivity of teaching. Interactive shaping can be seen as the problem of understanding *what people want* in a sequential decision-making context and of *how to provide these outcomes* that people desire.

Interactive shaping additionally accelerates learning, bringing the promise of assistive learning agents closer to fruition. Every human has vast knowledge about the world in which we exist; algorithms that learn from human-generated signals—like interactive shaping—will hasten robots' progress towards becoming worthy companions and collaborators.

Appendix A

Summary of the TAMER Framework for Interactive Shaping

This appendix contains a high-level overview of the TAMER framework for readers who wish to understand Chapters 4, 5, and 6 without reading Chapter 3, where TAMER is explained fully.

TAMER is a solution to the Interactive Shaping Problem, which asks how an agent can learn to perform a sequential task given only real-valued feedback on its actions from a human trainer. This problem is defined formally in Section 2.3. The human feedback, which we call "human reward",[1] is delivered through push buttons, spoken word, or any other easy-to-learn interface. The human's feedback is the *only* source of feedback or evaluation that the agent receives. However, solutions to the interactive shaping problem should be useful even when other evaluative information is available.

[1] Following common practice in reinforcement learning, we use "reward" to mean both positively and negatively valued feedback.

TAMER differs from past solutions to the Interactive Shaping Problem (or similar learning scenarios) in three important ways: (1) TAMER addresses delays in human evaluation through credit assignment, (2) TAMER learns a model of human reward ($\hat{R}_H$), and (3) at each time step, TAMER chooses the action that is predicted to directly elicit the maximum reward ($argmax_a \hat{R}_H(s, a)$), eschewing consideration of the action's effect on future state. TAMER is contrasted further with other approaches to interactive shaping in Section 2.5.1.

A.1 Motivation and philosophy of TAMER

The TAMER framework is designed around two insights. First, when a human trainer evaluates some behavior, she considers the long-term impact of that behavior, so her feedback signal contains her full judgement of the desirability of the targeted behavior. Second, a human trainer's feedback is only delayed by how long it takes to make and then communicate an evaluation. TAMER assumes that trainers' feedback is focused on recent behavior and not outcomes (which might be the result of any previous behavior or even independent of behavior); as a consequence, human reward is considered a trivially delayed, full judgment on the desirability of behavior.

Following the insights above and TAMER's assumption of behavior-focused feedback, TAMER avoids the credit assignment problem inherent in reinforcement learning. It instead treats human reward as fully informative about the quality recent actions from their corresponding states. Though the assumption of behavior-focused feedback may not always be true, the success of TAMER demonstrates that people can give reward that is compatible with the assumption. (Both instructions to the trainers and their interactions with the agents likely influence trainers' feedback.)

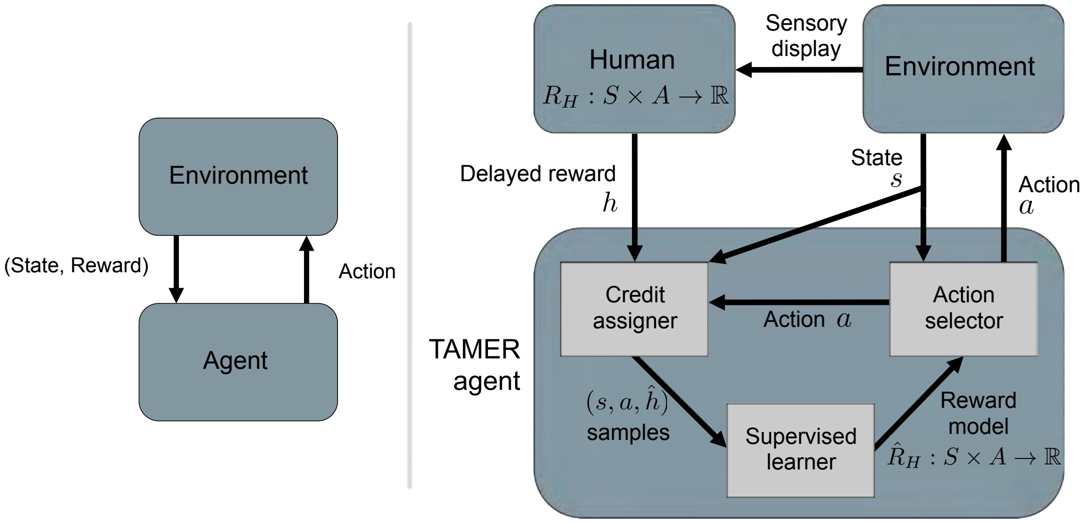


Figure A.1: On the left, the traditional scheme of interaction between agent and environment in a Markov Decision Process. On the right, the framework for Training an Agent Manually via Evaluative Reinforcement (TAMER). This figure is a repetition of Figure 3.1.

A.2 Mechanics of TAMER

The TAMER framework consists of three modules, as illustrated in Figure A.1: credit assignment to create labels from delayed reward signals for training samples, supervised learning from those samples to model human reward, and myopic action selection using the human reward model.

To model a hypothetical human reward function, $R_H : S \times A \to \mathbb{R}$, TAMER uses established regression algorithms to model this function; we call the model $\hat{R}_H$. Labels for state-action samples are constructed from real-valued human reward. The TAMER framework is agnostic to the specific model and supervised learner used, leaving such decisions to the agent's designer. However, we conjecture that the models should generalize well to unseen state-action pairs and weight recent training samples more highly, as the human's internal reward function is thought to change as the training session progresses.

In the full TAMER algorithm (described in Section 3.3.6), the small delay in addressed by spreading each human reward signal among multiple recent state-action pairs, contributing to the label of each. Section 3.3 describes this credit assignment method, including how each sample's weight is calculated from an estimated probability density function for the delay in reward delivery.

To choose actions within some state s, a TAMER agent directly exploits the learned model $\hat{R}_H$ and its predictions of expected reward. When acting greedily, a TAMER agent chooses the action $a = argmax_a[\hat{R}_H(s, a)]$. This is equivalent to performing reinforcement with a discount factor of 0, where reward acquired from future actions is not considered in action selection (hence the descriptor "myopic"). Other discounting rates are investigated in Chapter 6. In practice, almost all TAMER agents thus far have been greedy, since the trainer can punish the agent to make it try something different, making other forms of exploration less necessary. The one exception is in the feedback frequency experiment of Chapter 5, where non-greedy action occurs in certain conditions of the experiment and decreases learned performance of the agent.

Appendix B

Instructions given to subjects in interactive shaping user studies

In this appendix, we provide the full instructions given to subjects in the various user studies described throughout this thesis. Experiments for which the author was the trainer are omitted for lack of instructions.

B.1 First Tetris experiment in Chapter 3

We provided different instructions for the two groups of subjects: those with no computer science background and those who were PhD students or postdoctoral researchers in the Department of Computer Science at the University of Texas at Austin.

B.1.1 Instructions for subjects without computer science backgrounds

Any subject who did not have a computer science background was read instructions aloud, beginning with the following:

> In this experiment you will train a learning agent to play Tetris through

> positive and negative reinforcement. Practicing should take about 10 minutes. The real experiment will take about 15 minutes. For the time that any one Tetris piece is falling, you can give feedback about the previous Tetris piece's placement. So be careful not to give reinforcement for a move until it is fully completed. As the display will say, pressing the key 'p' gives positive reinforcement and pressing 'n' gives negative reinforcement. You can press either button multiple times to add strength to your feedback. You can also choose to not give feedback for any move. Ask me if you want to change the speed at which the game is played. You will have one practice period for getting used to operating the training system. Then there will be additional practice time after I read you some strategic suggestions.

The subject was then given a practice period of training the Tetris TAMER agent. Following this first practice session, further instructions were read aloud:

> As a trainer, your task is to let the Tetris player know which moves you approve and disapprove. Although you can use any strategy you like, we have two suggestions.
> 1) If the player has a pattern of behavior that you don't like, negatively reinforce anything related to that pattern.
> 2) If the player isn't doing something that you want it to, one strategy is to give negative reinforcement more often and positive reinforcement less often until you see the desired action. When you do see the desired action, we suggest heavily rewarding that action.
> Now you can practice as long as you want. If you want to practice with a fresh Tetris player, let me know and I'll restart it. Also, let me know when you are ready to start the real training session that will be included in my experimental data.

The subject then trained for a second practice session that lasted until they declared themselves ready to train in the session that counted in the study. The final training session lasted until the agent "lost" 10 episodes or the subject judged that they could not improve the agent any further and decided to end the training session and watch the agent play the rest of the 10 episodes at an increased speed.

B.1.2 Instructions for subjects with computer science backgrounds

The following instructions were sent by email. A "UCTS machine" is understood to mean a computer connected to the departmental network of the Department of Computer Science at the University of Texas at Austin.

> In this experiment you will train a learning agent to play Tetris to your preference. Starting it up and practicing should take as little as 5 minutes (more if you want to practice a lot). The real experiment will take about 15 minutes by my estimate.
>
> For the integrity of the data, please read this carefully.
>
> While sitting at a UTCS machine (don't use ssh -X unless you don't notice any lag):
>
> ```
> cd /projects/xxxx/rl-library/projects/experiments/guiExperiment/
> ```
>
> ```
> ./runNetDynamicEnvStandardAgent.bash
> ```
>
> In another, small window on the same machine:
>
> ```
> cd /u/xxxx/projects/shaping/agents/tetrisagent/
> ```
>
> ```
> ./run.bash -t yournamepractice
> ```
>
> (Of course, replace "yourname" with your actual name.)

To train a Tetris player:

- To start, in the RLVizApp window:

1. choose Tetris from the ”Choose Environment” drop-down box,

2. then click ”Load Experiment”,

3. then move the ”Simulation Speed” selector to around 150 (this can later be adjusted to your preference),

4. and then click ”Start”.

- As the agent plays, keep the second *terminal* window on top so that the agent can receive your keyboard-based feedback.

- ***** This part is easy to confuse. The window on top should be the terminal window with a box drawn around it. The text of the window starts ”Reward keys”. If you do not see the number after ”Human reinforcement for previous tetromino:” change when you give reward, the agent is not receiving your feedback. *****

- For the time that one Tetris piece is falling, you can give feedback about the previous piece placement. **Be careful that you don’t give feedback on a move until it is fully completed.**

- You can also choose to not give feedback for any move.

- Pressing ’p’ gives positive reinforcement and pressing ’n’ gives negative reinforcement. You can press either button multiple times to add strength to your feedback.

Play only long enough to get comfortable with the interface. Then restart the game for a fresh practice after reading the following strategy:

As a trainer, your task is to let the Tetris player know which moves you approve and disapprove. Although you can use any strategy you like, we have two suggestions:

1) If the player has a pattern of behavior that you don't like, negatively reinforce anything related to that pattern.

2) On the flipside, if the player isn't doing something that you want it to do, one strategy is to give negative reinforcement more often and positive reinforcement less often until you see the desired action. When you do, we suggest heavily rewarding that action.

At this point, you can practice as long as you want, restarting if you wish.

When you decide that you are ready to play for real (and for the prize!), restart the program, replacing "yournamepractice" in the terminal command with your name followed by the number 1 so that I know it's your first real training attempt. Your first attempt is the only one that I'll be using for sure, so do your best on this one. **An experimental run lasts 10 games.** Only at the end of the tenth game is any of your data saved to file. The current game number is printed above the Tetris board in the "E" part of "E/S/T". After the 10th game finishes, you can keep playing as long as you like, but no data will be recorded.

Just to be as clear as possible, I would type:

```
./run.bash -t me1
```

Whoever trains the best Tetris player on their first real run (no cheating or you'll be making me a fraud) wins their choice of some fine Russian vodka or some fine dining on me. The competition involves anyone from whom I collect data, including some non-CS people (who probably can't put up a fight). You can practice as much as you want beforehand, but once you do the yourname1 trial, please don't overwrite those data files. Runs are evaluated on the sum of reward received over the 10 games. Also, please *don't share results with each other* for the sake of the data's integrity. I'll announce the results after the data is collected.

If you need to take a break, you can click "Stop" in the RLVizApp window. When you want to start, click "Start" and then put the correct window on top (see the 5 star message above).

If you get bored because your agent is too good and never loses, you can speed it up and stop giving reinforcement.

A little warning: since I didn't master Python's Curses, the second terminal window will become pretty useless after the program ends. It's annoying. Sorry.

Thanks to everyone who helps.

B.2 Mountain car experiment in Chapter 3

This experiment also included both computer scientist subjects and non-technical subjects.

B.2.1 Instructions for subjects without computer science backgrounds

The following instructions were read aloud to subjects who did not have a background in computer science or artificial intelligence.

As a subject for this study, you will train a computer agent to complete a task. Specifically, the agent is a car that can choose to accelerate left, right, or not at all. The red rectangle represents the car, and the baby blue vertical bar on the car indicates the acceleration that is currently being done by the car. The task is for the car to get to the goal (the green marker on the top of the right hill) in the least time possible.

Your job will be to train the car agent to perform the task. As the trainer, you will give the agent positive and negative reinforcement as it explores different strategies. Your reinforcement will shape its behavior toward a strategy that efficiently makes it to the goal.

Before acting as a trainer, you will first learn, for yourself, a good strategy by controlling the car.

Keys 'S', 'D', and 'F' accelerate the car left, none, and right, respectively. You only have to push a key once for the car to keep choosing that action until either you press another key or the car reaches the goal.

Control the car until you think you have a clear idea of the fastest way to get to the goal. You will want to pay attention to when precisely the car should change its direction. (One of the biggest dangers to my results is that trainers might decide the agent is "good enough" when it could be much better. In this case, merely getting the car to the goal is not good enough.)

Note the red numbers above at the top of the display. The first number is the game number. The second is the measurement of how long the car the game has been going on (this is what you're trying to minimize.)

If you want to see it mimic your strategy, you can choose not give any

> commands at the start of a game (after the car reaches the goal). This is unrelated to any actual research.

The subject then controlled the mountain-car agent until it reached the goal a few times.

> Now you will train the car agent, giving it positive and negative reinforcement to shape its strategy into a good one. The input for this phase is ’p’ for positive reinforcement and ’n’ for negative reinforcement. In other words, push ’p’ immediately after behavior you approve of and push ’n’ after behavior you disapprove of. Use single button pushes for reinforcement (don’t hold it down).
>
> The agent learns best when reinforcement is both consistent and given very shortly after the action/event being reinforced.
>
> Train the agent for 20 episodes (i.e., it reaches the goal 20 times) or until you are confident that you cannot train the agent to improve its policy any further.
>
> Once you are done, speed up the agent and watch it for 50 episodes to make sure it doesn’t get stuck somewhere. If it does, give it negative reinforcement and watch it for another 50.
>
> You will train an agent 3 times this way. Expect to get better as a trainer as you progress (don’t get frustrated!).

> I’ll read a few notes before you begin. I can read them again at any time if you ask.
>
> - Up to a certain level, more frequent feedback generally makes for better learning.

- Be careful not to give feedback for something that hasn't happened yet. In other words, don't give reinforcement for an action that you anticipate but has not occurred.

At this point, the subject trained an agent in three separate sessions.

B.2.2 Instructions for those with computer science backgrounds

The instructions below were emailed to the subjects who were PhD students of postdoctoral researchers in the Department of Computer Science at the University of Texas at Austin. For this experiment, these technical subjects followed the instructions while the author was present for questions (though questions about the learning algorithm were not answered).

> As a subject for this study, you will train a computer agent to complete the "Mountain Car" task. Specifically, the agent is a car that can choose to accelerate left, right, or not at all. The task is for the car to get to the goal (a marker on the top of the right hill) in the least time possible.
>
> Your job will be to train the car agent to perform the task. As the trainer, you will give the agent positive and negative reinforcement as it explores different strategies. Your reinforcement will shape its behavior toward a strategy that efficiently makes it to the goal.
>
> The MDP environment and agent are started separately. Here's how to start the environment:
>
> Log in to a department computer and enter a GUI environment. (Don't use ssh, since it will cause a lag that might disrupt the results.)
>
> In a bash Terminal window, run:

export RLLIBRARY=/projects/xxxx/rl-library cd /projects/xxxx/rl-library/projects/experiments/guiExperiment ./runNetDynamicEnvStandardAgent.bash

It will then wait until it detects an agent.

Before acting as a trainer, you will first learn, for yourself, a good strategy by controlling the car.

To start a controllable agent, open another terminal window.

Give the following commands: export RLLIBRARY=/projects/xxxx/rl-library cd /u/xxxx/projects/shaping/agents/mcshapeagent ./runControl.bash -t yourname

This should connect an agent to the environment and bring up a GUI visualizer. Clicking the ”Load Experiment” button on the RLVizApp window. Then set the ”Simulation Speed” bar to 151 (any other speed will make the task harder or easier; sticking with 151 helps ensure the integrity of the results). Click ”Start”.

The red rectangle represents the car, and the baby blue vertical bar on the car indicates the acceleration that is currently being done by the car. The goal is the green marker on right hill.

For the agent to receive keyboard input, the most recently used terminal window has to be on top. Keys ’S’, ’D’, and ’F’ accelerate the car left, none, and right, respectively. You only have to push the key once for the car to keep choosing that action until another key is pressed or until the end of the episode.

Control the car until you think you have a clear idea of what a near-optimal policy (i.e., the fastest way to the goal) would look like. You will want to pay attention to when precisely the car should change its direction. (One of the biggest dangers to my results is that trainers might decide the agent is ”good enough” when it could be much better. In this case, merely getting the car to the goal is not good enough.)

Note the red numbers above at the top of the display. The first number is the game number. The second is the measurement of how long the car the game has been going on (this is what you’re trying to minimize.)

(If you don’t press anything at the beginning of the episode, an unrelated algorithm will take over until you exert control.)

——

[Start instruction loop (to be repeated 3 times)]

Now you will train the car agent, giving it positive and negative reinforcement to shape its strategy into a good one.

In the original window, press Ctrl-c to close the RL Visualizer application. You may have to do it twice. Issue command ./runNetDynamicEnvStandardAgent.bash again.

Close the newer window, which probably looks messed up now. Open a new window and run: export RLLIBRARY=/projects/xxxx/rl-library cd /u/xxxx/projects/shaping/agents/mcshapeagent ./runExperiment.bash -t yourname

You are about to start the environment again. Please read the rest of the instructions before you do, and then start training as quickly as possible after clicking ”Start” to avoid wasted time steps.

The input for this phase is ’p’ for positive reinforcement and ’n’ for negative reinforcement. In other words, push ’p’ immediately after behavior you approve of and push 'n' after behavior you disapprove of. Use single button pushes for reinforcement (don't hold it down).

** Strategic notes ** 1) The agent learns best when reinforcement is both consistent and given very shortly after the action/event being reinforced. 2) Up to a certain level, more frequent feedback generally makes for better learning. 3) Be careful not to give feedback for something that hasn't happened yet. In other words, don't give reinforcement for an action that you anticipate but has not occurred.

Train the agent for 20 episodes (i.e., it reaches the goal 20 times) or until you are confident that you cannot train the agent to improve its policy any further.

Once you are done, speed up the agent (set the speed bar to the far left) and watch it for 50 episodes to make sure it doesn't get stuck somewhere. If it does, quickly slow it down and give it negative reinforcement and watch it for another 50. Repeat until it does not get stuck (this won't take long).

You will start the environment the same way (same series of clicks within the RLVizApp window: "Load Experiment", speed bar to 151, and "Start"). Again, with the newer window open, you can give the agent keyboard input.

[End instruction loop] You will train an agent 3 times this way. Expect to get better as a trainer as you progress (don't get frustrated!).

B.3 The critique experiment from Chapter 5 (i.e., the second Tetris experiment in Chapter 3)

Below are the verbatim instructions given to participants from both the Teaching group and the Critiquing group from the critique experiment:

> **Teaching group:** You will be guiding a computer program that is trying to learn to play Tetris from scratch. You have the ability to reward the computer for making a good move or punish it for making a bad move. To reward it, press "?/" after the move is complete. To punish it, press "Z" after the move is complete. You can press the keys up to 4 times depending on how much you want to reward or punish it. Your evaluation will have a direct impact on how well the computer learns to play Tetris, so try to teach it as best you can. Think of this training as similar to training a dog. You would reward the dog when it performs well, and punish it when it performs poorly. Each game is over when the pieces reach the top.
>
> **Critiquing group:** You will be watching a playback recording of a computer program that tried to learn to play Tetris from scratch. Please evaluate this computer program by marking whether the computer made a good move or a bad move. If it has made a good move, press "?/" after the move is complete. If it has made a bad move, press "Z" after the move is complete. You can press the keys up to 4 times depending on how good or bad you think the move was. We are interested in how well you can evaluate computer programs that have tried to learn to play tetris. Each game is over when the pieces reach the top.

B.4 The feedback-frequency experiment from Chapter 5

In the feedback-frequency experiment, all participants received the same instructions:

> You will be guiding a computer program that is trying to learn to play Tetris from scratch. You have the ability to reward the computer for making a good move or punish it for making a bad move. If you feel it is necessary to reward it, press "?/" after the move is complete. If you feel it is necessary to punish it, press "Z" after the move is complete. You can press the keys up to 4 times depending on how much you want to reward or punish it. Your evaluation will have a direct impact on how well the computer learns to play Tetris, so try to teach it as best you can. Think of this training as similar to training a dog. You would reward the dog when it performs well, and punish it when it performs poorly. Give rewards or punishment whenever you feel it is necessary to do so. Each game is over when the pieces reach the top.

B.5 Discounting experiments in grid world in Chapter 6

These experiments were conducted through a web interface via Amazon.com's Mechanical Turk service. All interaction with an agent occurred through an applet that we built upon libraries within the RL Glue and RL Library projects [93]. The applet's display of the agent in its environment flashes a transparent blue when positive human reward is given and flashes a transparent red when negative human reward is given. Subjects received the following initial instructions in text:

> Kermitbot needs your help!
>
> Kermit the Robot Frog wants to be more like a real frog, but he doesn't understand that frogs need water. On top of that, he's always getting lost

> in the corridors. Poor Robo-kermie. Your job is to teach him how to go to the local watering hole. The videos below give the main instructions, but first:
>
> - The final result of your Robo-kermie pupil will be tested later for how quickly it can get to the water. The trainers of the best 50% of robots will be paid 25% more.
> - We strongly suggest closing all other applications currently running on your computer. Otherwise the game might have problems that hurt your chance of being in the top half of trainers.
> - Don't refresh your browser! If something is terribly wrong, describe it in detail at the end and you may receive credit.
> - Reasons we would reject your HIT: (1) You either did not watch the videos fully or did not answer the questions. (2) The records indicate that you did not honestly try to train the robot. (We will not reject simply for poor robot performance, though. But we will be sad.)
> - You can only do this HIT once. We will only pay each worker for one completion.
>
> Start your task here. You'll go through 6 steps. When you are asked for your HIT ID, enter *unique user ID* exactly (all the characters that are red above). Do not put in your worker ID!!! Once you finish, you'll be given a number. Then answer the questions below and enter the number at the bottom.

In accordance with the first bullet point above, subjects were paid either $0.80and1$ to participate in an approximately 18-minute experimental evaluation. The word "here" in the instructions above contains a hyperlink that opens a page with an introductory video. The script for the video is below, with on-screen text in brackets.

> [Teach a Kermitbot to find the water]
>
> For this Turk task, you'll be training a simulated robot to play a game. Together you'll form a team: you (the trainer) and the robot (the student). Your robot's performance in the game will be judged against that of other Turkers.
>
> Kermit the Robot Frog is very thirsty. He needs your help to find the water. So your goal is to teach the robot to find the water as fast as possible. [As fast as possible!]
>
> [Play the game yourself] Before you teach the robot, we'll have you do the task yourself by controlling it. Click on the box below, and Kermit to the water three times.

After watching the video, the subject controls the agent to get to the goal 3 times. The experiment will not progress if the subject does not complete this task. To the right of the applet containing the agent and its environment are the instructions, "To play the game, move Kermitbot to water with the arrow keys." Once this stage is complete, the subject clicks on a button to go to another page. Again, an instructional video is at the top; the script is below.

> Good job. Now I'll describe how you're going to train the agent.
>
> [Training basics] Here's the challenge. You'll be training the robot through reward and punishment signals. The forward slash button—which is also the question mark button—gives reward. Every time you push it, it gives a little bit of reward to the robot. You'll also see the screen flash blue when you give reward. The 'z' button gives punishment and will make the screen flash red. You can think of this as similar to training a dog or another animal through reward and punishment, but it will be somewhat different.

> [Pre-practice pointers] I'll give you a couple of pointers now before you practice.
>
> Number one. You can reward or punish rapidly to send a stronger signal than if you just pushed it once. [1. Rapid presses = stronger feedback] So, if I saw something I really liked, I might push reward, reward, reward, reward, reward, reward *(really fast, eventually inarticulate)* [well, not quite that much] and that would be a lot stronger than if I just pressed reward.
>
> Second pointer. The robot knows that people can only respond so quickly, so don't worry about your feedback being slightly delayed. [1. Rapid presses = stronger feedback, 2. Small delays in feedback are okay.]
>
> When you're ready to start practicing, click on the box below, and follow the instructions beside it.
>
> Try your hardest, but don't be hard on yourself. The first time is often rough, and this is just practice. Remember that, as the robot is learning from you, you are also learning, learning how to teach the robot.

As on the previous page, an applet is below the video. Through the applet, the subject practices training the agent until the agent reaches the goal twice or 200 time steps pass (at 800 milliseconds each). To the right of the applet are the following instructions:

> To train the robot:
> - '/' rewards recent behavior
> - 'z' punishes recent behavior
> - The arrow keys do nothing. Kermitbot is in control now.
>
> To control the game environment:
> - '0' pauses

- '2' unpauses

Once practice is complete, the subject clicks a button to move to another page. At the top is another video with the following script:

> [The real test] The mandatory practice is over, and the real test begins soon. If your practice training didn't go well, don't worry; it's just practice.
>
> [Closing instructions] For the real test, you'll train a little bit longer. The approximate amount of time is at the bottom of the screen. [3 minutes]
>
> Good luck, and thank you.

Through the applet below the video, the subject trains the agent in the session that is actually used as experimental data. To the right of the applet is the same set of instructions as on the previous page (indicating push-keys for training). The training session lasts for the durations reported in Sections 6.4.2 and 6.5.1, which differ for each of the two experiments. At the end of the training session, the trainer is given a "finish code" and is told to return to the original page. On this page is a questionnaire, which we use to screen for non-compliant subjects but have not analyzed further. At the end of the questionnaire, the trainer inputs the finish code received after training, completing their portion of the experiment.

Bibliography

[1] Abbeel, P., Coates, A., Quigley, M., Ng, A.: An application of reinforcement learning to acrobatic helicopter flight. Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference (2007)

[2] Abbeel, P., Ng, A.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the twenty-first international conference on Machine learning, p. 1. ACM (2004)

[3] Argall, B., Browning, B., Veloso, M.: Learning by demonstration with critique from a human teacher. In: Proceedings of the ACM/IEEE international conference on Human-robot interaction, pp. 57–64. ACM (2007)

[4] Argall, B., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robotics and Autonomous Systems **57**(5), 469–483 (2009)

[5] Bertsekas, D., Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scientific (1996)

[6] Bianchi, R., Ribeiro, C., Costa, A.: Heuristically Accelerated Q–Learning: a new approach to speed up Reinforcement Learning. Advances in AI – SBIA (2004)

[7] Bishop, C.: Pattern recognition and machine learning, vol. 4 (2006)

[8] Blumberg, B., Downie, M., Ivanov, Y., Berlin, M., Johnson, M., Tomlinson, B.: Integrated learning for interactive synthetic characters. Proc. of the 29th annual conference on Computer graphics and interactive techniques (2002)

[9] Bohm, N., Kokai, G., Mandl, S.: Evolving a heuristic function for the game of Tetris. Proc. Lernen, Wissensentdeckung und Adaptivitat LWA (2004)

[10] Boots, B., Siddiqi, S., Gordon, G.: Closing the learning-planning loop with predictive state representations. The International Journal of Robotics Research **30**(7), 954–966 (2011)

[11] Boumaza, A.: On the evolution of artificial tetris players. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on, pp. 387–393. IEEE (2009)

[12] Bouton, M.: Learning and Behavior: A Contemporary Synthesis. Sinauer Associates (2007)

[13] Boyan, J., Moore, A.: Generalization in reinforcement learning: Safely approximating the value function. Advances in neural information processing systems pp. 369–376 (1995)

[14] Breazeal, C.: Designing sociable robots. The MIT Press (2004)

[15] Breazeal, C.: Role of expressive behaviour for robots that learn from people. Philosophical Transactions of the Royal Society B: Biological Sciences **364**(1535), 3527–3538 (2009)

[16] Breazeal, C., Siegel, M., Berlin, M., Gray, J., Grupen, R., Deegan, P., Weber, J., Narendran, K., McBean, J.: Mobile, dexterous, social robots for mobile manipulation and human-robot interaction. SIGGRAPH08: ACM SIGGRAPH 2008 new tech demos (2008)

[17] Chen, D., Mooney, R.: Learning to interpret natural language navigation instructions from observations. Association for the Advancement of Artificial Intelligence (AAAI), Cambridge, MA (2011)

[18] Chernova, S., Veloso, M.: Interactive policy learning through confidence-based autonomy. Journal of Artificial Intelligence Research **34**(1), 1–25 (2009)

[19] Chernova, S., Veloso, M.: Teaching collaborative multi-robot tasks through demonstration. In: Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on, pp. 385–390. IEEE (2009)

[20] Cobo, L., Isbell Jr, C., Thomaz, A.: Automatic task decomposition and state abstraction from demonstration. Proceedings of The 11th Annual International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2012)

[21] Cobo, L., Zang, P., Isbell Jr, C., Thomaz, A.: Automatic state abstraction from demonstration. In: Twenty-Second International Joint Conference on Artificial Intelligence (2011)

[22] Dautenhahn, K.: Methodology and themes of human-robot interaction: a growing research field. International Journal of Advanced Robotic Systems **4**(1), 103–108 (2007)

[23] Dobbs, J., Arnold, D., Doctoroff, G.: Attention in the preschool classroom: The relationships among child gender, child misbehavior, and types of teacher attention. Early Child Development and Care **174**(3), 281–295 (2004)

[24] Dorigo, M., Colombetti, M.: Robot shaping: Developing situated agents through learning. Artificial Intelligence (1994)

[25] Evers, V., Maldonado, H., Brodecki, T., Hinds, P.: Relational vs. group self-construal: untangling the role of national culture in hri. In: Proceedings of

the 3rd ACM/IEEE international conference on Human robot interaction, pp. 255–262. ACM (2008)

[26] Fagot, B.: Influence of teacher behavior in the preschool. Developmental Psychology **9**(2), 198 (1973)

[27] Fahey, C.: Tetris ai, computer plays tetris (2003)

[28] Farias, V., Van Roy, B.: Tetris: A study of randomized constraint sampling. Probabilistic and Randomized Methods for Design Under Uncertainty pp. 189–201 (2006)

[29] Fernández, F., Veloso, M.: Probabilistic policy reuse in a reinforcement learning agent. AAMAS (2006)

[30] Gelly, S., Silver, D.: Achieving master level play in 9× 9 computer go. In: Proceedings of AAAI, pp. 1537–1540 (2008)

[31] Goldwasser, D., Roth, D.: Learning from natural instructions. In: Twenty-Second International Joint Conference on Artificial Intelligence (2011)

[32] Grollman, D., Jenkins, O.: Dogged learning for robots. In: Robotics and Automation, 2007 IEEE International Conference on, pp. 2483–2488. IEEE (2007)

[33] Hester, T., Stone, P.: Learning and using models. In: M. Wiering, M. van Otterlo (eds.) Reinforcement Learning: State of the Art. Springer Verlag, Berlin, Germany (2011)

[34] Hinds, P., Roberts, T., Jones, H.: Whose job is it anyway? a study of human-robot interaction in a collaborative task. Human-Computer Interaction **19**(1), 151–181 (2004)

[35] Hockley, W.E.: Analysis of response time distributions in the study of cognitive processes. Journal of experimental psychology. Learning, memory, and cognition **10** (1984)

[36] Isbell, C., Kearns, M., Singh, S., Shelton, C., Stone, P., Kormann, D.: Cobot in LambdaMOO: An Adaptive Social Statistics Agent. AAMAS (2006)

[37] Isbell, C., Shelton, C., Kearns, M., Singh, S., Stone, P.: A social reinforcement learning agent. In: Proceedings of the fifth international conference on Autonomous agents, pp. 377–384. ACM (2001)

[38] Judah, K., Roy, S., Fern, A., Dietterich, T.: Reinforcement Learning Via Practice and Critique Advice. AAAI (2010)

[39] Kaelbling, L., Littman, M., Moore, A.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research **4**, 237–285 (1996)

[40] Kakade, S.: A natural policy gradient. Advances in neural information processing systems **14**, 1531–1538 (2001)

[41] Kalyanakrishnan, S.: Learning methods for sequential decision making with imperfect representations. Ph.D. thesis, Department of Computer Science, The University of Texas at Austin, Austin, Texas, USA (2011). Published as UT Austin Computer Science Technical Report TR-11-41

[42] Kaochar, T., Peralta, R., Morrison, C., Fasel, I., Walsh, T., Cohen, P.: Towards understanding how humans teach robots. User Modeling, Adaption and Personalization pp. 347–352 (2011)

[43] Kaplan, F., Oudeyer, P., Kubinyi, E., Miklósi, A.: Robotic clicker training. Robotics and Autonomous Systems **38**(3-4) (2002)

[44] Kim, E., Leyzberg, D., Tsui, K., Scassellati, B.: How people talk when teaching a robot. In: Proceedings of the 4th ACM/IEEE international conference on Human robot interaction, pp. 23–30. ACM (2009)

[45] Kim, E., Scassellati, B.: Learning to refine behavior using prosodic feedback. In: Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on, pp. 205–210. IEEE (2007)

[46] Knox, W., Glass, B., Love, B., Maddox, W., Stone, P.: How humans teach agents: A new experimental perspective. International Journal of Social Robotics, Special Issue on Robot Learning from Demonstration (2012)

[47] Knox, W., Stone, P.: Interactively shaping agents via human reinforcement: The TAMER framework. The 5th International Conference on Knowledge Capture (2009)

[48] Knox, W., Stone, P.: Combining manual feedback with subsequent MDP reward signals for reinforcement learning. Proceedings of The 9th Annual International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2010)

[49] Knox, W.B., Breazeal, C., Stone, P.: Learning from feedback on actions past and intended. In: In Proceedings of 7th ACM/IEEE International Conference on Human-Robot Interaction, Late-Breaking Reports Session (HRI 2012) (2012)

[50] Knox, W.B., Stone, P.: Tamer: Training an agent manually via evaluative reinforcement. In: IEEE 7th International Conference on Development and Learning (2008)

[51] Knox, W.B., Stone, P.: Understanding human teaching modalities in reinforce-

ment learning environments: A preliminary report. In: IJCAI 2011 Workshop on Agents Learning Interactively from Human Teachers (ALIHT) (2011)

[52] Knox, W.B., Stone, P.: Reinforcement learning with human and MDP reward. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2012)

[53] Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. Machine Learning: ECML 2006 pp. 282–293 (2006)

[54] Kohl, N., Stone, P.: Policy gradient reinforcement learning for fast quadrupedal locomotion. In: IEEE International Conference on Robotics and Automation, vol. 3, pp. 2619–2624. Citeseer (2004)

[55] Konidaris, G., Niekum, S., Thomas, P.: TD$_\gamma$: Re-evaluating complex backups in temporal difference learning. In: Advances in Neural Information Processing Systems 24, pp. 2402–2410 (2011)

[56] Konidaris, G., Osentoski, S., Thomas, P.: Value function approximation in reinforcement learning using the Fourier basis. In: Proceedings of the Twenty-Fifth Conference on Artificial Intelligence, pp. 380–385 (2011)

[57] Kuhlmann, G., Stone, P., Mooney, R., Shavlik, J.: Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. In: The AAAI-2004 Workshop on Supervisory Control of Learning and Adaptive Systems (2004)

[58] Kuindersma, S.R., Hannigan, E., Ruiken, D., Grupen, R.A.: Dexterous mobility with the uBot-5 mobile manipulator. In: Proceedings of the 14th International Conference on Advanced Robotics. Munich, Germany (2009)

[59] Lagoudakis, M., Parr, R.: Least-squares policy iteration. The Journal of Machine Learning Research **4**, 1149 (2003)

[60] León, A., Morales, E., Altamirano, L., Ruiz, J.: Teaching a robot to perform task through imitation and on-line feedback. Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications pp. 549–556 (2011)

[61] Littman, M., Sutton, R., Singh, S.: Predictive representations of state. In: Advances in Neural Information Processing Systems (2002)

[62] MacDorman, K., Ishiguro, H.: The uncanny advantage of using androids in cognitive and social science research. Interaction Studies **7**(3), 297–337 (2006)

[63] MacDorman, K., Minato, T., Shimada, M., Itakura, S., Cowley, S., Ishiguro, H.: Assessing human likeness by eye contact in an android testbed. In: Proceedings of the XXVII annual meeting of the cognitive science society, pp. 21–23 (2005)

[64] Maclin, R., Shavlik, J.: Creating advice-taking reinforcement learners. Machine Learning **22**(1), 251–281 (1996)

[65] Mataric, M.: Reward functions for accelerated learning. ICML (1994)

[66] Menache, I., Mannor, S., Shimkin, N.: Basis function adaptation in temporal difference reinforcement learning. Annals of Operations Research **134**(1), 215–238 (2005)

[67] Mitchell, T.M.: Machine Learning. McGraw Hill (1997)

[68] Moreno, D., Regueiro, C., Iglesias, R., Barro, S.: Using prior knowledge to improve reinforcement learning in mobile robotics. Proc. Towards Autonomous Robotics Systems. Univ. of Essex, UK (2004)

[69] Nass, C., Reeves, B.: The media equation: How people treat computers, televisions, and new media as real people and places. Cambridge University Press (1996)

[70] Ng, A., Harada, D., Russell, S.: Policy invariance under reward transformations: Theory and application to reward shaping. ICML (1999)

[71] Nicolescu, M., Mataric, M.: Learning and interacting in human-robot domains. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on **31**(5), 419–430 (2002)

[72] Nicolescu, M., Mataric, M.: Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In: AAMAS, pp. 241–248. ACM (2003)

[73] Osentoski, S., Mahadevan, S.: Basis function construction for hierarchical reinforcement learning. In: AAMAS10: Proceedings of the 9th international joint conference on Autonomous agents and multiagent systems (2010)

[74] Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., Littman, M.: An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In: Proceedings of the 25th international conference on Machine learning, pp. 752–759. ACM (2008)

[75] Petrik, M., Scherrer, B.: Biasing approximate dynamic programming with a lower discount factor. Advances in neural information processing systems (2008)

[76] Pilarski, P., Dawson, M., Degris, T., Fahimi, F., Carey, J., Sutton, R.: Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In: Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on, pp. 1–7. IEEE (2011)

[77] Pomerleau, D.: ALVINN: An autonomous land vehicle in a neural network. In: Advances in Neural Information Processing Systems 1. Morgan Kaufmann (1989)

[78] Price, B., Boutilier, C.: Accelerating reinforcement learning through implicit imitation. JAIR **19**, 569–629 (2003)

[79] Pryor, K.: Don't shoot the dog!: the new art of teaching and training. Interpet Publishing (2002)

[80] Ramirez, K.: Animal training : successful animal management through positive reinforcement. Chicago, IL : Shedd Aquarium (1999)

[81] Ramon, J., Driessens, K.: On the numeric stability of gaussian processes regression for relational reinforcement learning. ICML-2004 Workshop on Relational Reinforcement Learning pp. 10–14 (2004)

[82] Randløv, J., Alstrøm, P.: Learning to drive a bicycle using reinforcement learning and shaping. In: Proceedings of the Fifteenth International Conference on Machine Learning, pp. 463–471. Citeseer (1998)

[83] Reed, K., Patton, J., Peshkin, M.: Replicating human-human physical interaction. In: IEEE International Conference on Robotics and Automation (2007)

[84] Rouder, J., Speckman, P., Sun, D., Morey, R., Iverson, G.: Bayesian t tests for accepting and rejecting the null hypothesis. Psychonomic Bulletin & Review **16**(2), 225–237 (2009)

[85] Saunders, J., Nehaniv, C., Dautenhahn, K.: Teaching robots by moulding behavior and scaffolding the environment. In: Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction, pp. 118–125. ACM (2006)

[86] Smart, W., Kaelbling, L.: Practical reinforcement learning in continuous spaces. ICML (2000)

[87] Sridharan, M.: Augmented reinforcement learning for interaction with non-expert humans in agent domains. In: Proceedings of IEEE International Conference on Machine Learning Applications (2011)

[88] Suay, H., Chernova, S.: Effect of human guidance and state space size on interactive reinforcement learning. In: RO-MAN, 2011 IEEE, pp. 1–6. IEEE (2011)

[89] Subramanian, K., Isbell, C., Thomaz, A.: Learning options through human interaction. In: 2011 IJCAI Workshop on Agents Learning Interactively from Human Teachers (ALIHT) (2011)

[90] Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press (1998)

[91] Szepesvári, C.: Algorithms for reinforcement learning. Synthesis Lectures on Artificial Intelligence and Machine Learning **4**(1), 1–103 (2010)

[92] Szita, I., Lorincz, A.: Learning Tetris Using the Noisy Cross-Entropy Method. Neural Computation **18**(12) (2006)

[93] Tanner, B., White, A.: RL-Glue : Language-independent software for reinforcement-learning experiments. Journal of Machine Learning Research **10**, 2133–2136 (2009)

[94] Tanner, B., White, A.: RL-Glue: Language-independent software for reinforcement-learning experiments. JMLR **10** (2009)

[95] Taylor, M., Suay, H., Chernova, S.: Integrating reinforcement learning with human demonstrations of varying ability. AAMAS (2011)

[96] Taylor, M.E.: Autonomous inter-task transfer in reinforcement learning do-

mains. Ph.D. thesis, Department of Computer Sciences, The University of Texas at Austin (2008)

[97] Taylor, M.E., Jong, N.K., Stone, P.: Transferring instances for model-based reinforcement learning. In: Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), pp. 488–505 (2008)

[98] Taylor, M.E., Stone, P.: Cross-domain transfer for reinforcement learning. In: Proceedings of the Twenty-Fourth International Conference on Machine Learning (2007)

[99] Taylor, M.E., Stone, P., Liu, Y.: Transfer learning via inter-task mappings for temporal difference learning. Journal of Machine Learning Research **8**(1), 2125–2167 (2007)

[100] Tellex, S., Kollar, T., Dickerson, S., Walter, M., Gopal Banerjee, A., Teller, S., Roy, N.: Approaching the symbol grounding problem with probabilistic graphical models. AI Magazine **32**(4), 64 (2011)

[101] Tenorio-Gonzalez, A., Morales, E., Villaseñor-Pineda, L.: Dynamic reward shaping: training a robot by voice. Advances in Artificial Intelligence–IBERAMIA 2010 pp. 483–492 (2010)

[102] Tesauro, G.: TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural computation **6**(2), 215–219 (1994)

[103] Thiery, C., Scherrer, B.: Improvements on learning tetris with cross entropy. International Computer Games Association Journal **32**(1), 311 (2009)

[104] Thomaz, A.: Socially guided machine learning. Ph.D. thesis, Citeseer (2006)

[105] Thomaz, A., Breazeal, C.: Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance. AAAI (2006)

[106] Thomaz, A., Breazeal, C.: Robot learning via socially guided exploration. In: Proceedings of the 6th International Conference on Development and Learning (2007)

[107] Thomaz, A., Breazeal, C.: Teachable robots: Understanding human teaching behavior to build more effective robot learners. Artificial Intelligence **172**(6-7), 716–737 (2008)

[108] Thomaz, A., Cakmak, M.: Learning about objects with human teachers. In: Proceedings of the 4th ACM/IEEE international conference on Human robot interaction, pp. 15–22. ACM (2009)

[109] Thomaz, A., Hoffman, G., Breazeal, C.: Reinforcement Learning with Human Teachers: Understanding How People Want to Teach Robots. In: ROMAN-06 (2006)

[110] Torres Peralta, R., Kaochar, T., Fasel, I., Morrison, C., Walsh, T., Cohen, P.: Challenges to decoding the intention behind natural instruction. In: RO-MAN, 2011 IEEE, pp. 113–118. IEEE (2011)

[111] Ugur, E., Celikkanat, H., Sahin, E., Nagai, Y., Oztop, E.: Learning to grasp with parental scaffolding. In: Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on, pp. 480–486. IEEE (2011)

[112] Watkins, C., Dayan, P.: Q-learning. Machine learning **8**(3), 279–292 (1992)

[113] Whitehead, S.: A study of cooperative mechanisms for faster reinforcement learning. University of Rochester, Dept. of Computer Science (1991)

[114] Wiewiora, E., Cottrell, G., Elkan, C.: Principled methods for advising reinforcement learning agents. ICML (2003)

[115] Wolfgang, C.: Solving discipline and classroom management problems: methods and models for today's teachers. Wiley (2004)

[116] Woodward, M., Wood, R.: Using bayesian inference to learn high-level tasks from a human teacher. In: International Conference on Artificial Intelligence and Pattern Recognition (AIPR-09) (2009)