Teaching on a Budget: Agents Advising Agents in Reinforcement Learning

Lisa Torrey
St. Lawrence University
ltorrey@stlawu.edu

Matthew E. Taylor
Washington State University
taylorm@eecs.wsu.edu

ABSTRACT

This paper introduces a teacher-student framework for reinforcement learning. In this framework, a teacher agent instructs a student agent by suggesting actions the student should take as it learns. However, the teacher may only give such advice a limited number of times. We present several novel algorithms that teachers can use to budget their advice effectively, and we evaluate them in two experimental domains: Mountain Car and Pac-Man. Our results show that the same amount of advice, given at different moments, can have different effects on student learning, and that teachers can significantly affect student learning even when students use different learning methods and state representations.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Intelligent agents

General Terms

Algorithms, Experimentation

Keywords

Reinforcement Learning; Agent Teaching; Advice Taking

1. INTRODUCTION

Using reinforcement learning (RL), agents can autonomously learn to master sequential-decision tasks. In these tasks, an agent must develop a control policy for taking actions in an environment. RL agents have traditionally been trained and used in isolation, but research is beginning to produce ways for them to interact productively with other agents and with humans.

This work focuses on how an RL agent could serve as a teacher for a task it has mastered. We begin with another RL agent in the role of the student, but we prefer teaching approaches that could potentially be used with human students as well. This limits us to human-understandable teaching methods, prevents teachers from assuming any access to a student's internal workings, and prevents students from simply starting with the teacher's knowledge. Furthermore, it requires teachers to be able to instruct students that may learn and perceive their environment differently.

As a motivating example for RL agents as teachers, consider the fast-growing industry of computer games. One measure of a successful game is how many humans learn to play it. Modern games often have built-in training sessions to help them; currently, this is additional content created by game developers. Instead, perhaps RL agents could learn to play these games autonomously and then teach human players. This could reduce the amount of developer time required to produce training content.

There are many possible ways to help agents learn [1, 13], but few are also applicable to human students. One that is applicable is *action advice*: as the student practices, the teacher suggests actions to take. We advocate this method because it requires minimal similarity between teachers and students — only a common action set. The agents may use different learning algorithms, and they may have different ways of representing the state of their environment. This is particularly important in light of the long-term goal of having human students, but it is also important to enable agents with different implementations (e.g., created by different companies) to teach each other without significant re-engineering.

Another assumption we make for this type of teaching is that teachers cannot give unlimited quantities of advice. Our primary reason for this restriction is that human students would have limited patience and attention. However, it is also true that some domains limit communication between agents. Furthermore, a teacher that over-advises a student could actually hinder its learning, if the differences between them are large enough.

This paper studies how an RL agent can best teach another RL agent using a limited amount of advice. The teacher observes the student and can give advice a fixed number of times, but cannot observe or change anything internal to the student. We propose a set of teaching algorithms: *early advising*, *importance advising*, *mistake correcting*, and *predictive advising*. We evaluate these algorithms experimentally in two domains: Mountain Car and Pac-Man. The results show that the same amount of advice, given at different moments, can have different effects on student learning, and that teachers can significantly affect student learning even when students use different learning methods and state representations.

2. REINFORCEMENT LEARNING

In reinforcement learning, an agent learns through trial and error to perform a task in an environment. As the agent takes actions, it receives feedback in the form of real-valued rewards. RL algorithms use this information to gradually improve an agent's control policy in order to maximize its total long-term expected reward.

At each step, the agent observes the state s of its environment. Using its policy π, it selects and performs an action a, which alters the environment state to s'. The agent observes this new state as well as a reward r, and it uses this information to update its policy. This cycle repeats throughout the learning process, which is often broken into a sequence of independent episodes.

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*
Copyright © 2013, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

A common way to represent a policy is with a Q-function $Q(s, a)$, which estimates the total reward an agent will earn starting by taking action a in state s. Given an accurate Q-function, the agent can maximize its rewards by choosing the action with the maximum Q-value in each state. Learning a policy therefore means updating the Q-function to make it more accurate. Even in the early stages of learning, the agent chooses actions with maximum Q-values most of the time, but to account for potential inaccuracies in the Q-function, it must perform occasional exploratory actions. A common strategy is ϵ-greedy exploration, where with a small probability ϵ, the agent chooses a random action.

In an environment with a reasonably small number of states, the Q-function can simply be a table of values with one entry for each state-action pair. Basic RL algorithms make updates to individual Q-value entries in this table. However, in some larger environments, states cannot be enumerated and need to be described by features $\{f_1, f_2, ...\}$. The Q-function is then an approximation, and a common form is a linear function $Q(s, a) = \sum_i w_i f_i$. Learning a policy then means updating the weights $\{w_1, w_2, ...\}$.

For the experiments in this paper, we use Q(λ) and Sarsa(λ) with linear Q-function approximation [12]. These are well-known RL algorithms that can incorporate advice with minimal modification. Since they already allow for off-policy exploratory actions, they can simply treat advice like a particularly lucky form of exploration. These algorithms have four parameters: the exploration rate ϵ, the learning rate α, the eligibility-trace parameter λ, and the discount factor γ. We report their values in each task for reproduceability but we omit a detailed discussion of them.

The weights $\{w_1, w_2, ...\}$ need to be given initial values. The usual choices are *optimistic*, so that weights are adjusted downwards over time, or *pessimistic*, so they are adjusted upwards. We find that this choice is important in the context of teaching with advice. With optimistic initialization, agents focus their attention on unexplored actions, which means that they delay repeating advised actions. With pessimistic initialization, agents have no such habit and can benefit much more from advice. The experiments in this paper therefore use pessimistic initialization.

3. TEACHING ON A BUDGET

Suppose that an RL agent has learned an effective policy π for a task. Using this fixed policy, it will teach another RL agent that is beginning to learn the same task. As the student learns, the teacher will observe each state s the student encounters and each action a the student takes. In n of these states, the teacher may advise the student to take the "correct" action $\pi(s)$.

How should the teacher spend its advice most effectively? Theoretically calculating this value is unlikely to be possible except in the simplest of RL problems. We instead take an experimental approach to this question, proposing and testing several algorithms for deciding when to give advice.

3.1 Early Advising

Students should benefit more from advice early on, when they know very little. Our first approach simply has the teacher give advice in the first n states the student encounters. This approach, which we call *early advising*, serves as our baseline.

```
procedure EarlyAdvising(π, n)
    for each student state s do
        if n > 0 then
            n ← n − 1
            Advise π(s)
```

3.2 Importance Advising

When all states in a task are equally important, early advising should be an effective strategy. However, we hypothesize that in some tasks, some states are more important than others, and saving advice for more important states would be a more effective strategy. Consider that games often have calmer and tenser moments. In certain situations, the right move can win the game or the wrong move can lose it; in others, any move is acceptable and none are disastrous. This is an intuitive definition of state importance, which we will soon quantify with a function $I(s)$.

A teacher that is conscious of state importance could give advice only when it reaches some threshold t. We call this approach *importance advising*.

```
procedure ImportanceAdvising(π, n, t)
    for each student state s do
        if n > 0 and I(s) ≥ t then
            n ← n − 1
            Advise π(s)
```

When t is 0, this becomes equivalent to early advising, assuming importance values are non-negative.

Because our teachers are RL agents with Q-functions, they have a natural way to calculate $I(s)$. Recall that a Q-value $Q(s, a)$ is an estimate of the rewards ultimately achievable by taking action a in state s. If the Q-values for all the actions in s are the same, then it does not matter which one is taken, and s is unimportant. However, if some actions in s have larger Q-values than others, then it does matter, and s has some importance. For this paper, we therefore define state importance as:

$$I(s) = \max_a Q(s, a) - \min_a Q(s, a)$$

This measure was introduced by Clouse [3] in his work on apprenticeship learning, but it was used there to approximate a learner's confidence in a state. Here, we compute $I(s)$ with the teacher's fully-learned Q-function rather than the student's partially-learned one, and in this context it is a better indicator of state importance than agent confidence.

3.3 Mistake Correcting

Even if a teacher saves its advice for important states, it may end up wasting some advice in states where the student had already intended to take the correct action. Advice can only have an effect when used in states where the student makes mistakes. If teachers could restrict their advice to these states, they should be able to improve upon both of the above methods.

However, one of our key assumptions in this work is that teachers have no direct access to student knowledge. To make it possible for teachers to spend advice exclusively on mistakes, students would need to announce their intended actions in advance and give teachers an opportunity to correct them. This introduces additional communication into the framework that may not be convenient in all situations, but the approach serves as a useful upper bound. We call this approach *mistake correcting*.

```
procedure MistakeCorrecting(π, n, t)
    for each student state s do
        Observe student's announced action a
        if n > 0 and I(s) ≥ t and a ≠ π(s) then
            n ← n − 1
            Advise π(s)
```

When t is 0, this approach ignores state importance and corrects all mistakes.

3.4 Predictive Advising

Although teachers cannot directly access student knowledge, they may be able to infer students' policies from their behavior. A teacher observes the states a student encounters and the actions it takes. Using these observations as training data, the teacher can train a classifier to predict student actions, and use these predictions in place of student announcements. We call this approach *predictive advising*.

procedure PREDICTIVEADVISING(π, n, t)
  for each student state s **do**
    Predict student's intended action a
    if $n > 0$ and $I(s) \geq t$ and $a \neq \pi(s)$ **then**
      $n \leftarrow n - 1$
      Advise $\pi(s)$

This approach approximates mistake correcting, but has the advantage of not requiring additional communication from the student. If a teacher's action predictor performs perfectly, predictive advising becomes equivalent to mistake correcting. When it makes inaccurate predictions, the teacher sometimes wastes advice, making predictive advising more like importance advising. Inaccurate predictions can also make the teacher miss opportunities to give useful advice. Although other such opportunities may arise later, the delay of useful advice could make predictive advising perform worse than importance advising.

Many algorithms for supervised learning could potentially be applied to this classification task. In this paper we use a Support Vector Machine, as implemented in the *SVM-Light* software package [4]. The details of the SVMs depend upon the teacher's state representation and are discussed in the sections describing our experimental domains, but in general, the SVMs map state features (as the teacher sees them) to student actions.

Each student state-action pair generates one training example. The teacher trains a new SVM after each episode using training examples from the previous episode. Average SVM training times are approximately one second. This is an inconspicuous delay between episodes, but it could be disruptive during episodes, which is why we do not update the SVMs more often.

Note that this classification task is inherently challenging for several reasons. First, students are constantly learning and will likely produce inconsistent training data because their behavior is nonstationary. Second, students sometimes take random exploration steps, which means the data will be noisy. Third, the student's state representation can be different from the teacher's, which means the hypothesis space of the classifier may not even contain the student's policy. Despite these challenges, our results will show that useful prediction is achievable in some scenarios.

4. EXPERIMENTAL DOMAINS

We evaluate these four teaching algorithms in two experimental domains. Mountain Car is a well-understood benchmark domain and Pac-Man is a complex stochastic game.

4.1 Mountain Car

In the Mountain Car task [10], the agent is a car in the two-dimensional environment pictured in Figure 1 (left). There are three actions the agent can take: accelerate in the $+x$ direction, accelerate in the $-x$ direction, or do not accelerate at all. The effects of these actions are determined by a simple physics simulation. The car begins each episode with zero velocity at the bottom of the mountain, and the episode ends when it reaches the goal at the top, or after 500 steps. The car's motor is underpowered so that

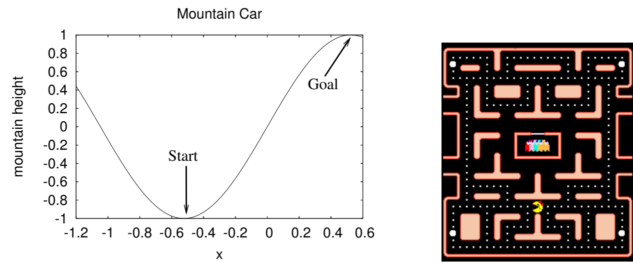


Figure 1: Left: the Mountain Car slope, where x is the car's location. Right: the Pac-Man maze.

it can only reach the top by oscillating back and forth to build momentum. The learner receives a -1 reward at every step until the episode ends.

States in this domain are points in a space of two dimensions: the car's position x and velocity v. This continuous space is usually discretized with *tile coding* [12]. A $k \times k$ grid is laid across it, and each square tile in the grid produces a feature whose value is 1 if (x, v) falls within that tile and 0 otherwise. There are m such tilings, each offset by a different amount, so the state representation consists of mk^2 tile features. To perform linear Q-function approximation with tile coding, each action associates a different weight with each tile. A Q-value $Q(s, a)$ is the sum of action a's weights for the m active tiles in state s.

To perform experiments in which the student and teacher use different state representations, we vary k and m. We select two tile codings in which both Q(λ) and Sarsa(λ) succeed: one coarser with $k = 8, m = 16$ and one finer with $k = 16, m = 8$. Both allow agents to achieve an average reward of roughly -150, but the first setting reaches this asymptotic performance in about 100 episodes, while the second takes about 300 episodes.

Mountain car is not a task in which single moves can "win" or "lose" an episode, but its states do have a range of importance. Figure 2 (left) shows how $I(s)$ varies in a typical episode experienced by a coarse-coding Sarsa(λ) teacher. Each teacher has its own range for $I(s)$, but the oscillating pattern of rising importance occurs for all teachers. This indicates that importance-based advising may be applicable in this domain.

Action prediction in Mountain Car is a straightforward mapping of a state (mk^2 tile features) to one of the three actions. This is done with a multi-class version of SVM-Light using a linear kernel. We keep most of the SVM-Light parameters at their default values, but the margin/error tradeoff C is tuned to 10000 by having teachers predict their own actions.

These tuning experiments also provide an indication of how well teachers should be able to predict student actions in this domain. Unfortunately, at best, the prediction accuracy is roughly 50%. While this is better than random guessing (33%), we do not expect predictive advising to approximate mistake correcting under these conditions. The reason for this low accuracy is likely that the number of features (1024-2048) is large compared to the number of training examples (150-500). Providing more training data would normally improve performance, but not necessarily when the data is nonstationary, as student behavior is. This tension between the need for more data and the need for recent data makes action prediction difficult in this domain.

4.2 Pac-Man

Pac-Man is a 1980s arcade game in which the player navigates a maze like the one in Figure 1 (right), trying to earn points by

touching edible items and trying to avoid being caught by the four ghosts. We use an implementation of the game provided by the Ms Pac-Man vs. Ghosts League [9], which conducts annual competitions. Ghosts in this implementation chase the player 80% of the time and move randomly the other 20%.

Pac-Man episodes all occur in the same maze. The agent has four actions — move up, down, left, and right — but in most states only some of these actions are available. Four moves are required to travel between the small dots on the grid, which represent food pellets and are worth 10 points each. The larger dots are power pellets, which are worth 50 points each, and also cause the ghosts to become edible for a short time, during which they slow down and turn to fleeing instead of chasing. Eating a ghost is worth 200 points and causes the ghost to respawn in the lair at the center of the maze. The episode ends if any ghost catches Pac-Man, or after 2000 steps.

This domain is discrete but has a very large state space. There are 1293 distinct locations in the maze, and a complete state consists of the locations of Pac-Man, the ghosts, the food pellets, and the power pills, along with each ghost's previous move and whether or not it is edible. The combinatorial explosion of possible states makes it essential to approach this domain through high-level feature construction and Q-function approximation.

Useful high-level features tend to describe distances between Pac-Man and other objects of interest. Action-specific features are more useful than global features. For example, a global feature might be "the distance from Pac-Man to the nearest food pellet." Making this feature specific to action a, it becomes "the distance from Pac-Man to the nearest food pellet after Pac-Man executes a."

When using action-specific features, a feature set is really a set of functions $\{f_1(s,a), f_2(s,a), ...\}$. All actions share one Q-function, which associates a weight with each feature. A Q-value is $Q(s,a) = w_0 + \sum_i w_i f_i(s,a)$. To achieve gradient-descent convergence, it is important to have the extra bias weight w_0 and also to normalize the features to the range $[0, 1]$.

We create agents with different state representations in this domain by defining two distinct feature sets. One feature set consists of 16 features that count objects at a range of distances from Pac-Man. The other consists of 7 heavily-engineered distance-related features. These features are not fully documented here for space reasons, but their implementation is available upon request.

A perfect score in an episode would be 5600 points, but this is quite difficult to achieve (for both humans and agents). An agent executing random actions earns an average of 250 points. The 16-feature set allows an agent to reach an average of 2600 points per episode, successfully eating most of the pellets and the occasional edible ghost. The 7-feature set allows an agent to learn to catch more edible ghosts and achieve a per-episode average of 3800 points. We therefore refer to the 16-feature set as "low-asymptote" and the 7-feature set as "high-asymptote."

In Pac-Man, some moves can lose the game or achieve large rewards, while others are largely trivial. Figure 2 (right) shows how $I(s)$ varies in a typical episode experienced by a high-asymptote Sarsa(λ) teacher. The periodic peaks indicate that importance-based advising should be applicable in this domain.

Action prediction must work differently in Pac-Man than in Mountain Car because all Pac-Man states are not represented the same way. Different subsets of actions are available in different states and the features are action-specific, so not all states have the same number of features. This rules out the simple approach of mapping from features to actions. Instead, we approach action prediction in Pac-Man as a *ranking* problem, which can be solved using another version of SVM-Light.

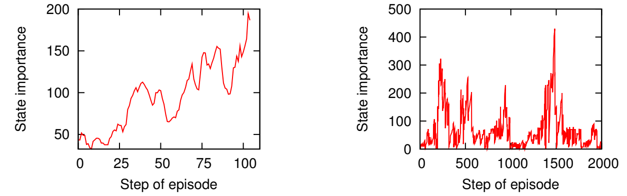


Figure 2: Trained agents in Mountain Car (left) and Pac-Man (right) measuring state importance during a typical episode

Consider a teacher using the 7-feature set, observing a state s in which the actions *up*, *down*, and *left* are available, and in which the student chooses to move *up*. The corresponding training example generated for the ranking SVM would be structured like this:

2 $f_1(s,\text{up})$, $f_2(s,\text{up})$, ..., $f_7(s,\text{up})$
1 $f_1(s,\text{down})$, $f_2(s,\text{down})$, ..., $f_7(s,\text{down})$
1 $f_1(s,\text{left})$, $f_2(s,\text{left})$, ..., $f_7(s,\text{left})$

In this example, each action is represented by one line of features. The numbers in the left column specify pairwise ranking constraints between actions. Since *up* has a higher number than *down* and *left*, the SVM applies the constraints $Q(s,\text{up}) > Q(s,\text{down})$ and $Q(s,\text{up}) > Q(s,\text{left})$. Since *down* and *left* have the same numbers, no constraints are generated between them. This reflects the teacher's knowledge, which is only that the student chose *up* over the other actions.

The output of the ranking SVM, when queried on a state s, is a set of real numbers, one for each action available in s. The predicted student action in s is the one with the highest number. We again keep most of the SVM-Light parameters at their default values, except for the margin/error tradeoff C, which is tuned to 1000. Prediction accuracy in these tuning experiments was consistently in the 80-90% range. A likely factor in this result is the small ratio of features to training examples. We expect predictive advising to be more effective in this domain, relative to Mountain Car.

5. TEACHING RESULTS

This section demonstrates improvements in student learning via teaching in the Mountain Car and Pac-Man tasks. First, agents are trained independently with all combinations of learning algorithms and state representations. For each combination, we select the best-performing agent to be a teacher. These teachers then give advice to students of several combinations.

To smooth the natural variance in student performance, each learning curve averages at least 30 independent trials of student learning. While training, an agent pauses every few episodes to perform at least 30 evaluation episodes and record its average performance. No learning, exploration, or teaching takes place during evaluation episodes. This way the learning curves only reflect student knowledge, not teacher knowledge.

One learning curve is better than another if it has a steeper slope or a higher asymptote. Agents that use different learning algorithms, state representations, or parameter settings can differ in both of these ways. Our experiments focus on the impact of teaching when these other factors are fixed.

We do not expect teaching to change the asymptotic performance of students, but we do look for it to improve their learning speed. To measure learning speed in a holistic way, we compute areas under learning curves. Curves are compared using t-tests on their areas with $\alpha = 0.05$. When we report that the difference between two curves is significant, it means we have at least 95% confidence that one curve has larger area.

With the exception of early advising, all of our teaching approaches have a parameter t, the threshold above which a state is considered important. To explore how t affects performance, we try 10 values for each teacher, uniformly distributed across that teacher's $I(s)$ range. For each teacher-student pair, we report the most effective value for t.

5.1 Pac-Man

Pac-Man teachers are given an advice budget of $n = 1000$, which is roughly half the number of steps in a single well-played episode. The RL parameters that agents use are $\epsilon = 0.05$, $\alpha = 0.001$, $\gamma = 0.999$, and $\lambda = 0.9$.

First, we present experiments where the teacher and student use the same algorithm and feature set (see Figure 3). On the left of this figure, both agents use Sarsa(λ) and the low-asymptote 16-feature set; $t = 50$. Differences between curves are significant for all pairs except mistake correcting and predictive advising. On the right, both agents use Sarsa(λ) and the high-asymptote 7-feature set; $t = 200$. Differences between curves are significant for all pairs except early advising and independent learning.

Although these teachers are giving advice in only a small fraction of the training steps, some of them have significant effects on student learning. Advice has a higher overall impact on students with the 16-feature set because these students have a simpler policy to learn. Early advising provides a large benefit with the 16-feature students, but not with the 7-feature ones. Importance advising is slightly but consistently better than early advising, and the best t thresholds are above 0, which confirms that saving advice for important states can be effective. Mistake correcting consistently outperforms importance advising, which confirms that saving advice for mistakes is also effective. Predictive advising also outperforms importance advising, and for the 16-feature students it even matches mistake correcting. These results suggest that mistake correcting is the best choice if it is feasible, and if not, predictive advising is the best alternative.

The good performance of predictive advising on the left of Figure 3 is not due to perfect action prediction. In fact, prediction accuracy is lower on the left (79%) than on the right (86%). But prediction errors are less costly when teaching low-asymptote students because they benefit more from any advice schedule. For the same student, increased prediction accuracy corresponds to better performance with predictive advising. However, the impacts of wasted and delayed advice are not the same for all students.

Our next experiments investigate the effects of having the teacher and student use different learning algorithms. This factor would be irrelevant if all algorithms converged to the same optimal policy, but in practice this is not the case. Q(λ) and Sarsa(λ) produce asymptotic policies for Pac-Man whose performances differ by approximately 200 points. They also learn at different speeds; for the sake of variety, we exaggerate this difference by using $\lambda = 0.7$ in Q(λ) to slow it further.

In Figure 4, on the left, a Q(λ) teacher advises a Sarsa(λ) student; $t = 20$ and prediction accuracy is 80%. Differences between curves are significant for all pairs except early advising and importance advising. On the right, the algorithms are reversed, $t = 100$, and prediction accuracy is 81%. Differences between curves are significant for all pairs except mistake correcting and predictive advising. All of these agents use the 16-feature set.

Although these students use different algorithms and progress at different rates than their teachers, the differences do not appear to hinder teaching. Advice has a higher overall impact on students that use Sarsa(λ), probably because advice is treated as exploration, and Sarsa(λ) takes exploration into account more than Q(λ) does. These results suggest that the learning algorithm of the student has more impact on the effectiveness of teaching than the learning algorithm of the teacher does. While some students may respond better to advice than others, teachers can effectively advise students that learn differently.

Finally, we present experiments where the teacher and student use different feature sets. We expect this to be the most challenging type of difference because it causes large differences in the asymptotic performance of teachers and students. It is not obvious that advice will be helpful across this divide, and there is even the risk that it might be harmful.

In Figure 5, on the left, a high-asymptote 7-feature teacher advises a low-asymptote 16-feature student; $t = 100$. Differences between curves are significant for all pairs. On the right, the feature sets are reversed and $t = 250$. Mistake correcting and predictive advising are significantly different from each other and the rest, but the other three approaches are statistically equivalent. All of these agents use Sarsa(λ).

Although these teachers perceive their environment differently than their students, some of them still provide significant benefits on student learning. These effects are not always as strong as when teachers and students used the same state representation, but some of them remain quite useful. Unsurprisingly, high-asymptote teachers have larger effects on low-asymptote students than vice versa. But low-asymptote teachers do have positive impacts on high-asymptote students (with mistake correcting and predictive advising), and these students then go on to outperform their teachers, as they should given their higher inherent capability. None of these teachers have negative impacts on students.

The high-asymptote agents have substantial difficulty predicting the actions of the low-asymptote agents (accuracy 61%). This causes predictive advising to perform slightly below importance advising on the left of Figure 5. There is no such difficulty in the reverse scenario (accuracy 86%). Prediction accuracy across feature sets is likely to depend on the specifics of the features. However, these results suggest that teachers can effectively advise students that perceive the world differently.

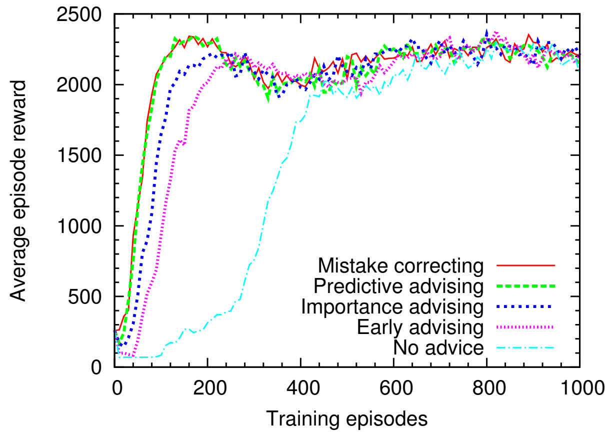
The rate at which teachers spend their advice is partly controlled by the importance threshold t. When teaching 16-feature students, the best teachers give most of their advice within the first 10 episodes of student training, because the low-asymptote students benefit most from advice very early in their learning. When 16-feature teachers advise 7-feature students, they also do best to spend their advice quickly, before the high-asymptote students surpass them. However, when 7-feature teachers advise 7-feature students, they can perform better by giving less frequent advice over longer periods. The best settings of importance advising, mistake correcting and predictive advising spread their advice over 20, 100 and 60 episodes respectively.

5.2 Mountain Car

Mountain Car is easier to learn independently, and has weaker state-importance contrasts and action prediction compared to Pac-Man. Because of these factors, we expect our teaching methods to have less variation and less overall impact in this domain. In particular, although we include predictive advising, we do not expect it to be a close approximation of mistake correcting.

Mountain Car teachers are given an advice budget of $n = 100$, which again is less than the number of steps in a well-played episode. The RL parameters are $\epsilon = 0.05$, $\alpha = 0.08$, $\gamma = 1.0$, and $\lambda = 0.9$.

First, we present experiments where the teacher and student use the same algorithm and tile coding (see Figure 6). On the left of this figure, both agents use Sarsa(λ) and the finer coding; $t = 175$



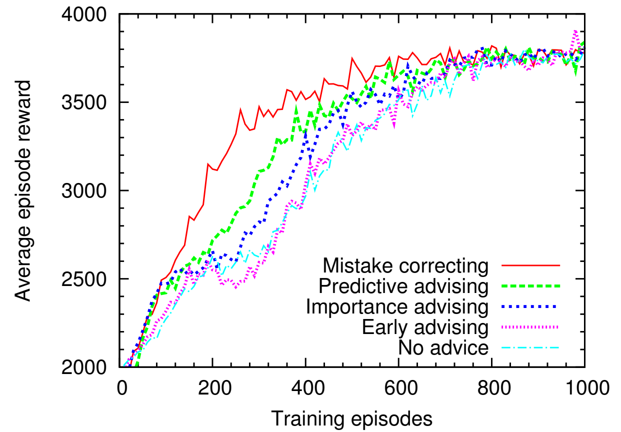
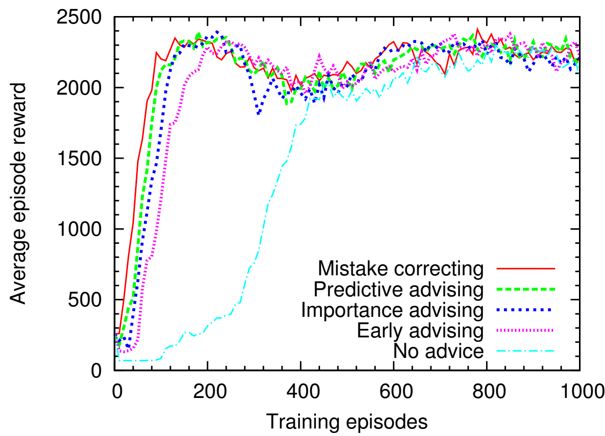


Figure 3: Pac-Man teaching with similar students and teachers using Sarsa. Left: low-asymptote feature set. Right: high-asymptote feature set.



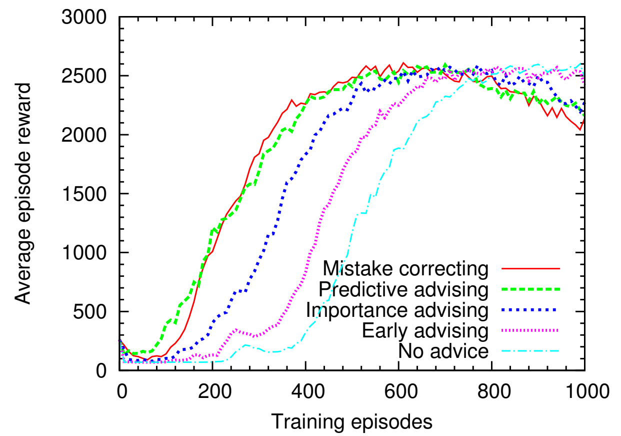
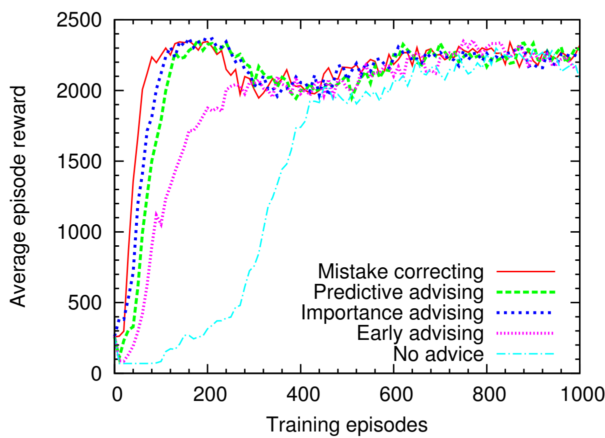


Figure 4: Pac-Man teaching with different learning algorithms using the low-asymptote feature set. Left: Q-Learning teacher and Sarsa student. Right: Sarsa teacher and Q-Learning student.



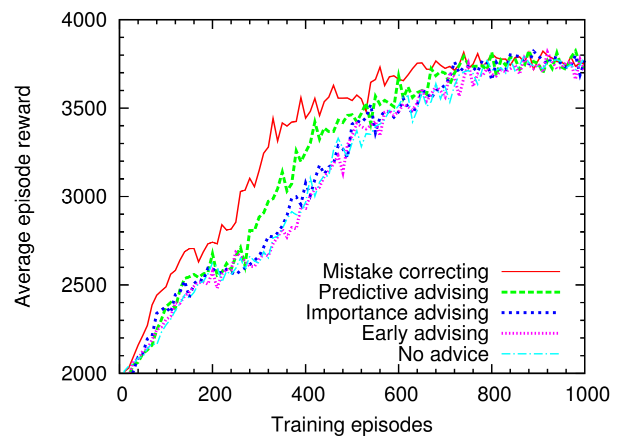
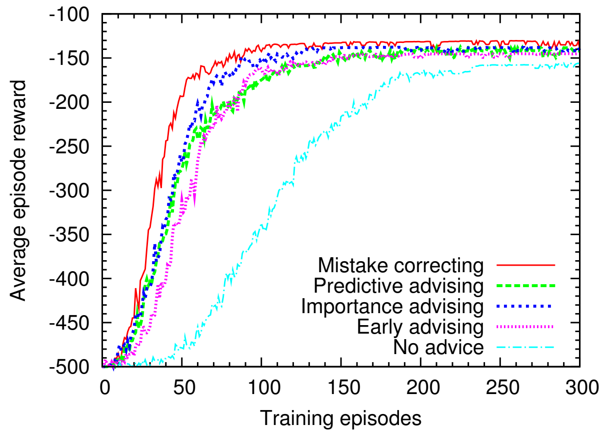


Figure 5: Pac-Man teaching with different feature sets using Sarsa. Left: high-asymptote teacher with low-asymptote student. Right: low-asymptote teacher with high-asymptote student.



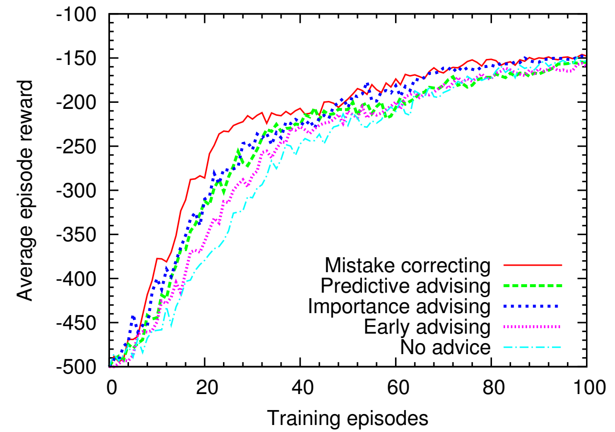
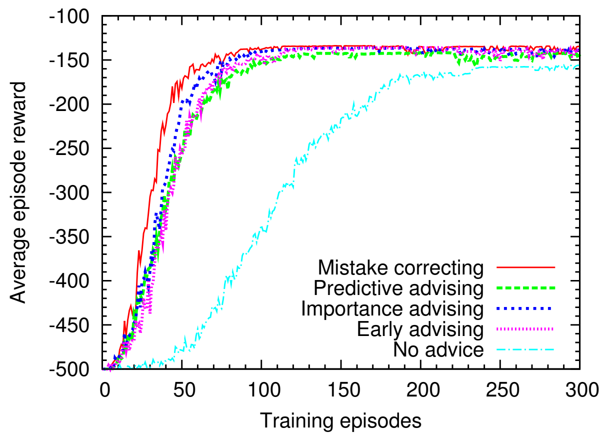


Figure 6: Mountain Car teaching with similar students and teachers. Left: fine coding with Sarsa. Right: coarse coding with Q-Learning.



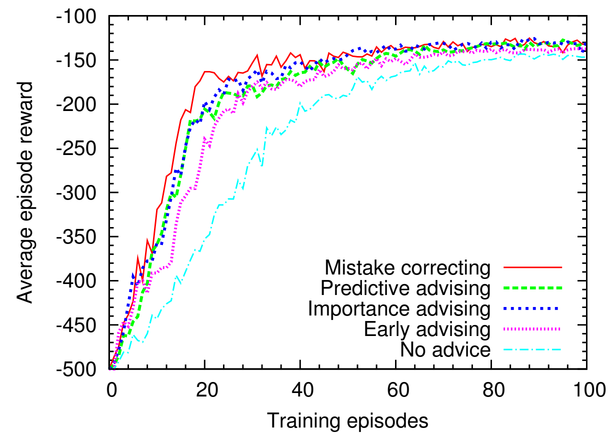


Figure 7: Mountain Car teaching with different tile codings using Sarsa. Left: coarse-coding teachers with fine-coding students. Right: fine-coding teachers with coarse-coding students.

and prediction accuracy is 45%. Differences between curves are significant for all pairs except predictive advising and early advising. On the right, both agents use Q(λ) and the coarser coding; $t = 180$ and prediction accuracy is 52%. Mistake correcting and importance advising are significantly different from the rest, but the other three approaches are not statistically different.

Although these teachers are giving small amounts of advice, some of them do improve student learning. Mistake correcting and importance advising tend to have small advantages over the other teaching algorithms. This indicates that saving advice for important states and for mistakes are still useful heuristics in this domain. Predictive advising, as expected, no longer dominates importance advising.

As in Pac-Man, Sarsa(λ) students benefit more from advice than Q(λ) students. Fine-coding students also benefit more from advice than coarse-coding students; they have a larger space to explore, and advice helps them discover important areas more quickly. Both of these effects make the impact of teaching higher on the left of Figure 6 than on the right, but they are observable separately in other experiments that we omit due to space constraints. Similarly omitted are experiments showing that as in Pac-Man, it is not harmful for teachers and students to use different learning algorithms.

In our final experiments, the teacher and student use different tile codings. We expect this difference to be less challenging than the difference between state representations in Pac-Man, because two reasonable tile codings are more similar than two reasonable feature sets.

In Figure 7, on the left, a coarse-coding teacher advises a fine-coding student; $t = 120$ and prediction accuracy is 59%. Differences between curves are significant for all pairs except predictive advising and early advising. On the right, the tile codings are reversed, $t = 175$, and prediction accuracy is 44%. Differences between curves are significant for all pairs except importance advising and predictive advising. All of these agents use Sarsa(λ).

Although these teachers represent states differently than their students do, teaching is just as effective as when they use the same tile coding. The relative performance of the teaching algorithms follows the same pattern. Overall, these results indicate that our algorithms can allow teachers to effectively advise students with different tile codings.

As in Pac-Man, the importance threshold t controls how quickly teachers spend their advice. The best teachers for coarse-coding students give their advice more quickly (within about 10 episodes) than the best teachers for fine-coding students (within about 20

episodes), because the former learn more quickly than the latter and get more benefit from earlier advice.

5.3 Results Summary

Our experimental results lead us to the following conclusions about teaching with an advice budget.

1. Student learning can be improved with a small advice budget.
2. Advice can have greater impact when it is spent on more important states.
3. Advice can have greater impact when it is spent on mistakes.
4. When teachers can successfully predict student mistakes, they can spend their advice budget more effectively.
5. Teaching can improve student learning even when agents have different learning algorithms or state representations.
6. Students can benefit from advice even from teachers with less inherent ability, and then go on to outperform their teachers.

6. RELATED WORK

There are several types of related work in the area of helping agents to learn. Some of this work involves teaching in non-RL settings, such as classification [2], or involves collaborative teams of RL agents [11]. These areas of research address the same high-level goal of productive agent interaction, but their problem settings are somewhat different.

In the field of *transfer learning* in RL [13], an agent uses knowledge from a source task to aid its learning in a target task. However, agents performing transfer often have direct access to source-task knowledge. In contrast, our work assumes student agents have strict limits on access to teacher knowledge.

More closely related work has one RL agent teach another without a direct knowledge transfer. For example, in *experience replay* [5], a student trains on the recorded experiences of a teacher. This requires the student to have an identical state representation, which is a limitation our methods avoid. Other examples include *imitation learning* [8], in which a student learns by observing a teacher, *apprentice learning* [3], in which a student asks a teacher for advice whenever its confidence in a state is low, and *advice exchange* [7], in which peers ask for advice from each other based on heuristics of self-confidence and trust. Our work diverges from these by having an expert teacher decide when to give advice, and by focusing on the effective use of small advice amounts.

Finally, humans are sometimes employed to teach agents. *Learning from Demonstration* [1] includes a broad category of work that focuses on agents learning to mimic a human demonstrator. There has also been work on allowing humans to communicate rules expressing their knowledge of a domain [6]. Research in these areas tends to focus on compensating for human error, which is not an important issue in our work with agent teachers.

7. CONCLUSIONS

As more problems become solvable by agent-based methods, it is important for agents to be able to work together, even if they are implemented differently. It is also important for agents and humans to be able to interact productively despite their substantial differences. RL agents are good at learning control policies for specific tasks, and it would be useful for them to be able to serve as teachers for those tasks.

This paper poses the problem of having trained RL agents serve as teachers in ways that are effective for many types of students. We present teaching algorithms that use small amounts of action advice to speed up student learning, even when students learn and represent states differently. Our experimental results show that significant benefits, as measured by areas under student learning curves, are achievable with these algorithms.

There are many potential directions for future work. For example, action prediction might be improved by using an efficient incremental classifier. The concept of state importance could also use further exploration: perhaps there exist better domain-specific ways to measure state importance, or effective strategies for automatically selecting and adjusting importance thresholds.

Larger steps in future work could extend the problem to include multiple teachers and/or students. It would also be useful to examine agents with broader ranges of learning algorithms, including human students. Domains like Pac-Man would be particularly suitable for these kinds of experiments. We hope that this study will provide a foundation for additional work in this area.

8. ACKNOWLEDGEMENTS

We thank Chris HolmesParker for helpful comments and suggestions. This work was supported in part by NSF IIS-1149917.

9. REFERENCES

[1] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483, 2009.

[2] D. Chakraborty and S. Sen. Teaching new teammates. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 691–693, 2006.

[3] J. A. Clouse. *On integrating apprentice learning and reinforcement learning*. PhD thesis, University of Massachusetts, 1996.

[4] T. Joachims. Making large-scale SVM learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

[5] L. J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[6] R. Maclin and J. W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22(1-3), 1996.

[7] L. Nunes and E. Oliveira. On learning by exchanging advice. *AISB Journal*, 1(3), 2003.

[8] B. Price and C. Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.

[9] P. Rohlfshagen and S. M. Lucas. Ms Pac-Man versus Ghost Team CEC 2011 competition. In *Proceedings of the 2011 IEEE Congress on Evolutionary Computation*, CEC'11, pages 70–77, 2011.

[10] S. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22, 1996.

[11] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence*, July 2010.

[12] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[13] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.