EXERCISES 23

REINFORCEMENT LEARNING

23.1 Learning from Rewards

Exercise 23.RLAN

Investigate the application of reinforcement learning ideas to the modeling of human and animal behavior.

Reinforcement learning as a general "setting" can be applied to almost any agent in any environment. The only requirement is that there be a distinguished reward signal. Thus, given the signals of pain, pleasure, hunger, and so on, we can map human learning directly into reinforcement learning—although this says nothing about how the "program" is implemented. This view misses out several important issues. First, we do not know what the human "reward function" is and how it relates to perceptual experience. Second, there are other forms of learning that occur in humans. These include "speedup learning" (Chapter 22); supervised learning from other humans, where the teacher's feedback is taken as a distinguished input; and the process of learning the world model, which is "supervised" by the environment.

Exercise 23.RLEV

Is reinforcement learning an appropriate abstract model for evolution? What connection exists, if any, between hardwired reward signals and evolutionary fitness?

To map evolutionary processes onto the formal model of reinforcement learning, one must find evolutionary analogs for the reward signal, learning process, and the learned policy. Let us start with a simple animal that does not learn during its own lifetime. This animal's genotype, to the extent that it determines animal's behavior over its lifetime, can be thought of as the parameters θ of a policy pi_θ. Mutations, crossover, and related processes are the part of the learning algorithm—like an empirical gradient neighborhood generator in policy search—that creates new values of θ. One can also imagine a reinforcement learning process that works on many different copies of π simultaneously, as evolution does; evolution adds the complication that each copy of π modifies the environment for other copies of π, whereas in RL the environment dynamics are assumed fixed, independent of the policy chosen by the agent. The most difficult issue, as the question indicates, is the reward function and the underlying objective function of the learning process. In RL, the objective function is to find policies that maximize the expected sum of rewards over time. Biologists usually talk about evolution as maximizing "reproductive fitness," i.e., the ability of individuals of

a given genotype to reproduce and thereby propagate the genotype to the next generation. In this simple view, evolution's "objective function" is to find the π that generates the most copies of itself over infinite time. Thus, the "reward signal" is positive for creation of new individuals; death, *per se*, seems to be irrelevant.

Of course, the real story is much more complex. Natural selection operates not just at the genotype level but also at the level of individual genes and groups of genes; the environment is certainly multiagent rather than single-agent; and, as noted in the case of Baldwinian evolution in Chapter 4, evolution may result in organisms that have hardwired reward signals that are related to the fitness reward and may use those signals to learn during their lifetimes.

As far as we know there has been no careful and philosophically valid attempt to map evolution onto the formal model of reinforcement learning; any such attempt must be careful not to *assume* that such a mapping is possible or to *ascribe* a goal to evolution; at best, one may be able to interpret what evolution tends to do *as if* it were the result of some maximizing process, and ask what it is that is being maximized.

23.2 Passive Reinforcement Learning

Exercise 23.PASL

Implement a passive learning agent in a simple environment, such as the 4×3 world. For the case of an initially unknown environment model, compare the learning performance of the direct utility estimation, TD, and ADP algorithms. Do the comparison for the optimal policy and for several random policies. For which do the utility estimates converge faster? What happens when the size of the environment is increased? (Try environments with and without obstacles.)

The code repository shows an example of this, implemented in the passive 4×3 environment. The agents are found under `lisp/learning/agents/passive*.lisp` and the environment is in `lisp/learning/domains/4x3-passive-mdp.lisp`. (The MDP is converted to a full-blown environment using the function `mdp->environment` which can be found in `lisp/uncertainty/environments/mdp.lisp`.)

Exercise 23.PRPO

Chapter 16 defined a **proper policy** for an MDP as one that is guaranteed to reach a terminal state. Show that it is possible for a passive ADP agent to learn a transition model for which its policy π is improper even if π is proper for the true MDP; with such models, the POLICY-EVALUATION step may fail if $\gamma = 1$. Show that this problem cannot arise if POLICY-EVALUATION is applied to the learned model only at the end of a trial.

Consider a world with two states, S_0 and S_1, with two actions in each state: stay still or move to the other state. Assume the move action is non-deterministic—it sometimes fails, leaving the agent in the same state. Furthermore, assume the agent starts in S_0 and that S_1 is a terminal state. If the agent tries several move actions and they all fail, the agent may conclude

that $T(S_0, Move, S_1)$ is 0, and thus may choose a policy with $\pi(S_0) = Stay$, which is an improper policy. If we wait until the agent reaches S_1 before updating, we won't fall victim to this problem.

Exercise 23.PRSW

Starting with the passive ADP agent, modify it to use an approximate ADP algorithm as discussed in the text. Do this in two steps:

a. Implement a priority queue for adjustments to the utility estimates. Whenever a state is adjusted, all of its predecessors also become candidates for adjustment and should be added to the queue. The queue is initialized with the state from which the most recent transition took place. Allow only a fixed number of adjustments.

b. Experiment with various heuristics for ordering the priority queue, examining their effect on learning rates and computation time.

This question essentially asks for a reimplementation of a general scheme for asynchronous dynamic programming of which the prioritized sweeping algorithm is an example (Moore and Atkeson, 1993). For **a.**, there is code for a priority queue in both the Lisp and Python code repositories. So most of the work is the experimentation called for in **b**.

Exercise 23.DIRU

The direct utility estimation method in Section 23.2 uses distinguished terminal states to indicate the end of a trial. How could it be modified for environments with discounted rewards and no terminal states?

When there are no terminal states there are no sequences, so we need to define sequences. We can do that in several ways. First, if rewards are sparse, we can treat any state with a reward as the end of a sequence. We can use equation (21.2); the only problem is that we don't know the exact totl reward of the state at the end of the sequence, because it is not a terminal state. We can estimate it using the current $U(s)$ estimate. Another option is to arbitrarily limit sequences to n states, and then consider the next n states, etc. A variation on this is to use a sliding window of states, so that the first sequence is states $1 \ldots n$, the second sequence is $2 \ldots n+1$, etc.

23.3 Active Reinforcement Learning

[[need exercises]]

23.4 Generalization in Reinforcement Learning

Exercise 23.TDLQ

Write out the parameter update equations for TD learning with

$$\hat{U}(x, y) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 \sqrt{(x - x_g)^2 + (y - y_g)^2} \, .$$

This utility estimation function is similar to equation (21.9), but adds a term to represent Euclidean distance on a grid. Using equation (21.10), the update equations are the same for θ_0 through θ_2, and the new parameter θ_3 can be calculated by taking the derivative with respect to θ_3:

$$\begin{aligned}
\theta_0 &\quad \theta_0 + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) , \\
\theta_1 &\quad \theta_1 + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) x , \\
\theta_2 &\quad \theta_2 + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) y , \\
\theta_3 &\quad \theta_3 + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) \sqrt{(x - x_g)^2 + (y - y_g)^2} \, .
\end{aligned}$$

Exercise 23.VCRL

Adapt the vacuum world (Chapter 2) for reinforcement learning by including rewards for squares being clean. Make the world observable by providing suitable percepts. Now experiment with different reinforcement learning agents. Is function approximation necessary for success? What sort of approximator works for this application?

The conversion of the vacuum world problem specification into an environment suitable for reinforcement learning can be done by merging elements of `mdp->environment` from `lisp/uncertainty/environments/mdp.lisp` with elements of the corresponding function `problem->environment` from `lisp/search/agents.lisp`. The point here is twofold: first, that the reinforcement learning algorithms in Chapter 21 are limited to accessible environments; second, that the dirt makes a huge difference to the size of the state space. Without dirt, the state space is $O(n)$ with n locations. With dirt, the state space is $O(2^n)$ because each location can be clean or dirty. For this reason, input generalization is clearly necessary for n above about 10. This illustrates the misleading nature of "navigation" domains in which the state space is proportional to the "physical size" of the environment. "Real-world" environments typically have some combinatorial structure that results in exponential growth.

Exercise 23.APLM

Implement an exploring reinforcement learning agent that uses direct utility estimation. Make two versions—one with a tabular representation and one using the function approxi-

mator in Equation (23.9). Compare their performance in three environments:

a. The 4×3 world described in the chapter.

b. A 10×10 world with no obstacles and a +1 reward at (10,10).

c. A 10×10 world with no obstacles and a +1 reward at (5,5).

Code not shown. Several reinforcement learning agents are given in the directory `lisp/learning/agents`.

Exercise 23.RLGR

Devise suitable features for reinforcement learning in stochastic grid worlds (generalizations of the 4×3 world) that contain multiple obstacles and multiple terminal states with rewards of $+1$ or -1.

Possible features include:

- Distance to the nearest $+1$ terminal state.
- Distance to the nearest -1 terminal state.
- Number of adjacent $+1$ terminal states.
- Number of adjacent -1 terminal states.
- Number of adjacent obstacles.
- Number of obstacles that intersect with a path to the nearest $+1$ terminal state.

Exercise 23.RLGM

Extend the standard game-playing environment (Chapter 6) to incorporate a reward signal. Put two reinforcement learning agents into the environment (they may, of course, share the agent program) and have them play against each other. Apply the generalized TD update rule (Equation (23.11)) to update the evaluation function. You might wish to start with a simple linear weighted evaluation function and a simple game, such as tic-tac-toe.

The modification involves combining elements of the environment converter for games (`game->environment` in `lisp/search/games.lisp`) with elements of the function `mdp->environment`. The reward signal is just the utility of winning/drawing/losing and occurs only at the end of the game. The evaluation function used by each agent is the utility function it learns through the TD process. It is important to keep the TD learning process (which is entirely independent of the fact that a game is being played) distinct from the game-playing algorithm. Using the evaluation function with a deep search is probably better because it will help the agents to focus on relevant portions of the search space by improving the quality of play. There is, however, a tradeoff: the deeper the search, the more computer time is used in playing each training game.

Exercise 23.TENX

Compute the true utility function and the best linear approximation in x and y (as in Equation (23.9)) for the following environments:

a. A 10×10 world with a single $+1$ terminal state at (10,10).

b. As in (a), but add a -1 terminal state at (10,1).

c. As in (b), but add obstacles in 10 randomly selected squares.

d. As in (b), but place a wall stretching from (5,2) to (5,9).

e. As in (a), but with the terminal state at (5,5).

The actions are deterministic moves in the four directions. In each case, compare the results using three-dimensional plots. For each environment, propose additional features (besides x and y) that would improve the approximation and show the results.

This is a relatively time-consuming exercise. Code not shown to compute three-dimensional plots. The utility functions are:

a. $U(x, y) = 1 - \gamma((10 - x) + (10 - y))$ is the true utility, and is linear.

b. Same as in a, except that $U(10, 1) = -1$.

c. The exact utility depends on the exact placement of the obstacles. The best approximation is the same as in a. The features in exercise 21.9 might improve the approximation.

d. The optimal policy is to head straight for the goal from any point on the right side of the wall, and to head for (5, 10) first (and then for the goal) from any point on the left of the wall. Thus, the exact utility function is:

$$\begin{aligned} U(x, y) &= 1 - \gamma((10 - x) + (10 - y)) \qquad (\text{if } x \geq 5) \\ &= 1 - \gamma((5 - x) + (10 - y)) - 5\gamma \qquad (\text{if } x < 5) \end{aligned}$$

Unfortunately, this is not linear in x and y, as stated. Fortunately, we can restate the optimal policy as "head straight up to row 10 first, then head right until column 10." This gives us the same exact utility as in a, and the same linear approximation.

e. $U(x, y) = 1 - \gamma(|5 - x| + |5 - y|)$ is the true utility. This is also not linear in x and y, because of the absolute value signs. All can be fixed by introducing the features $|5 - x|$ and $|5 - y|$.

23.5 Policy Search

[[need exercises]]

23.6 Apprenticeship and Inverse Reinforcement Learning

[[need exercises]]

23.7 Applications of Reinforcement Learning

[[need exercises]]