

Assigned: Tuesday 10 December 2013

Due: Friday 20 December 2011, 4:45 pm

Objectives:

- Understand details of **model-free** passive reinforcement learning
- Build an active, **exploratory Q-learning** agent
- Empirically test **agent performance**

Collaboration: This homework assignment will be completed in pairs assigned by the instructor.

Introduction

We conclude our study of learning for Markov decision processes by letting our passive agent forgo a world transition model, directly estimating utilities under a fixed policy. Finally, our most general agent will forgo the world model and learn a policy directly through exploration.

Code

You will need `mdp.c` and `environment.c` from the prior labs. You may obtain the starter code for this assignment on the MathLAN from the directory

```
~weinman/courses/CSC261/code/tdq
```

Here is an overview of the files it contains.

`max.c` Contains functions for finding the largest value in an array or the index of the largest value in an array over a restricted set of indices (useful for the max/argmax operations in Q-learning).

`td.c` Contains a skeleton for setting up a temporal-difference learning agent.

`qlearn.c` Contains a skeleton for setting up an active Q-learning agent.

Assignment

In addition to those listed below, 12 points will be allotted for general good programming practice.

Problem 1: Temporal-difference learning [16 points]

The file `td.c` contains a setup for running an agent that directly learns utilities (using no world transition model) in a sequential environment for a specified number of iterations. Implement the function `rl_agent_action` according to PASSIVE-TD-AGENT of AIMA Figure 21.4 (p. 837).

At the top of the file you will notice several persistent variables that are initialized for you. To assist you, a summary of the conversions from the pseudo code to C statements are given below.

Pseudo-Code	C
current state s' and reward signal r'	parameters <code>state</code> and <code>reward</code>
s, a, r the previous state, action, and reward	<code>prevState, prevAction, prevReward</code>
$s, a, r \leftarrow \text{null}$	<code>prevValid = 0</code>
$U[s']$	<code>utilities[state]</code>
$N_s[s]$	<code>state_freq[prevState]</code>
$\pi[s']$	<code>policy[state]</code>
$\alpha(n)$	<code>double updateWeight(double freq)</code>

Note that the only environment property available to your agent (via `p_mdp`) is now the terminal indication. Like before, the `Makefile` contains commands to build your program with the command

```
$ make td
```

Problem 2: Q-Learning [Extra Credit: 16 points]

The file `qlearn.c` contains a setup for running an agent that directly learns a policy (using no utilities or transition model) in a sequential environment for a specified number of iterations. The Q-LEARNING-AGENT algorithm below gives a pseudo-code implementation appropriate for our environment. In particular, in contrast with the AIMA implementation (Figure 21.8) note that our Q table has no slot for “no available action”, *None*. To compensate for this, we set *all* Q values for a state to the reward, and store the maximum Q value accordingly.

Algorithm 1 An exploratory Q-learning agent.

```
function Q-LEARNING-AGENT( percept ) returns an action
  inputs: percept, a percept indicating the current state  $s'$  and reward signal,  $r'$ 
  persistent:  $Q$ , a table of action values indexed by state and action, initially zero
              $N_{sa}$ , a table of frequencies for state-action pairs, initially zero
              $s, a, r$  the previous state, action, and reward, initially null
  local variables:  $Q^*$ , the maximizing Q-value for the current state

  if TERMINAL?( $s'$ ) then
    for each action  $a'$  do
       $Q[s', a'] \leftarrow r'$ 
     $Q^* \leftarrow r'$ 
  else
     $Q^* \leftarrow \max_{a' \in A(s')} Q[s', a']$ 
  if  $s$  is not null then
    increment  $N_{sa}[s, a]$ 
     $Q[s, a] \leftarrow Q[s, a] + \alpha (N_{sa}[s, a]) (r + \gamma Q^* - Q[s, a])$ 
  if TERMINAL?( $s'$ ) then
     $s, a, r \leftarrow \text{null}$ 
  else
     $s, a, r \leftarrow s', \arg \max_{a' \in A(s')} f(Q[s', a'], N_{sa}[s', a']), r'$ 
  return  $a$ 
```

Implement the `rl_agent_action` function of `qlearn.c` using Q-LEARNING-AGENT of the algorithm above as a guide. Some important notes:

- As in prior examples, the persistent variables are declared and initialized for you. Namely
 - $N_{sa}[s, a]$ is `state_action_freq[prevState][prevAction]` and
 - $Q[s', a']$ is `state_action_value[state][action]`.

- $f(u, n)$ is implemented by

```
double exploration_function( double u, double n )
```

using the form on p. 842 with R^+ given by the global variable `bestReward` and N_e given by the global variable `minTries`, both of which are command-line arguments (`reward` and `attempts`, respectively).

- You can use `max_value` (from `max.h`) to determine Q^* .

- You probably *cannot* use `arg_max_value` to determine the best f value. Be careful to only evaluate f using the available actions for a state, which is what $a' \in A(s')$ means.

The Makefile contains commands to build your program with the command

```
$ make qllearn
```

Problem 3: Application [32 points]

Complete, compiled versions of `td` and `qllearn` are included with the original files copied from the MathLAN; you may use them or your own versions for experimentation.

Part A: Temporal Difference Learning

Test Test the passive, model-free TD-learner using the (presumably optimal) policy learned by policy iteration. For example,

```
$ ./policy_iteration gamma epsilon 4x3.mdp > 4x3.policy
$ ./td gamma epsilon 4x3.mdp trials < 4x3.policy
```

How many trials does it take before your utilities match those of Figure 21.1(b) to at least two decimals?

Predict Before running the TD-learner on the larger 16×4 world, record how many trials do you expect your agent to require to estimate state utilities accurately? Why?

Experiment Run `td` on the 16×4 grid world using the (presumably optimal) policy learned by policy iteration and the same discount factor and error tolerance you used for the 4×3 grid world. How many trials does it take for the learned utilities to match those from your prior lab as closely as possible?

Reflect Compare your predictions to your experimental outcome. Speculate about the cause of any discrepancies.

Part B: Q-Learning

Test Test the active, model-free Q-function learner on the 4×3 grid world. What values for the optimistic best *reward* (R^+) and *attempts* (N_e) are required to find the most optimal policy at a given number of trials? Why

Predict Before running your Q-learning agent on the larger 16×4 world, what values of *reward* (R^+), *attempts* (N_e), and trials do you expect your agent to require to learn the best possible policy. Why?

Experiment Run your `qllearn` on the 16×4 grid world using the same discount factor you used for the 4×3 grid world. What settings are required for your agent to learn the best policy?

Reflect Compare your predictions to your experimental outcome. Speculate about the cause of any discrepancies.

What to turn in

Your submission should include the following

- Your completed `qllearn.c` and `tdp.c`
- A single PDF containing (merged)
 - Your C files (Note that your `enscript` command should use the flag `-Ec` rather than `-Escheme`)

- A transcript of your `tdp` and `qlearn` programs' compilations (be sure to make `clean` first) and output for both grid worlds
- Responses to Problem 3

Submissions missing the PDF or files in any other formats will not be graded. C files lacking transcripts will not be graded.

