Assignment 2, Part II

Introduction

In this assignment you will implement a simple Q-learning algorithm to solve a particular task: balancing a pole on a cart.

The Pole Balancing Problem

The pole balancing problem is a classic one in reinforcement learning. In it, the controller is attempting to keep a pole that is balanced on a cart from falling flat by applying pushes to the cart. Pole balancing code can be found on the internet at http://www-anw.cs.umass.edu/rlr/domains.html

If you wish you can use this code as a starting point/reference, or write your own in C/C++ or Java.

The Q Representation

The game version of this problem should have five possible actions – push slight left, push left, push slight right, push right, do nothing. You then need to figure out how to represent the current pole balancing situation as a state. You should consider representing the state of the system by looking at two or three values about the cart and pole:

• The angle of the pole: since this is a continuous value you should divide the possible angles into a set of discrete bins. For example, break the angle values into groups like -90 degrees to -30 degrees (bin 0), -30 degrees to -20 degrees (bin 1), -20 to -10 degrees (bin 2), -10 degrees to -5 degrees (bin 3), -5 degrees to 0 (bin 4), etc.
• The angular velocity of the pole: this is also a continuous value. And again you should pick an appropriate set of bins.
• Since the game should also terminate when the cart runs into the left or right wall, you may want to add something to the state indicating when the cart is close to either wall (e.g., three bins, one when the cart is near the left wall, one when it is near the right wall, and one when it is not near either).

If your representation has 10 bins for angle, 8 for angular velocity, and 3 for the cart position then there are 240 possible states. You can give each state a
unique number by calculating a value like this:
State =
(\text{ANGLEBIN} - 1) \times 8 \times 3 + (\text{VELOCITYBIN} - 1) \times 3 + (\text{POSITION} - 1)

Then your Q table is simply a two dimensional array with the first dimension being the number of states and the second dimension the number of actions.

**Problem 1: Learning the Q Table**

To learn the Q table you should run a large number of pole balancing games. A game should end either when the pole drops or if 500 steps are reached. The reward the controller receives is a large negative value when the pole drops or when the cart hits the wall. All other rewards should be 0.

For the discount factor I would suggest a high value such as 0.9 or 0.95 (you may want to make this an input to your system). To select from amongst the actions I would suggest an approach involving a probability of selecting the "best" (highest Q value action). With some probability \( p \) at each step you should choose the best action, otherwise choose an action at random. To make this technique work I would start this probability at a low value during early learning (allowing lots of exploration) and then increasing the probability for later games (to allow more exploitation). However, feel free to experiment with smarter methods.

Your general learning approach should work as follows:

```
FOR game = 1 TO MaxNumberOfTrainingGames DO
    initialize a pole balancing game
    REPEAT
        determine the state
        select an action
        perform that action
        measure the reward
        update the Q function
    UNTIL the pole drops OR 500 steps have happened
    IF a certain number of games have passed THEN
        evaluate the current Q table
```
To evaluate the current Q table you should run a certain number of games (perhaps 50) and count for each game how many steps occur before the pole drops (or that 500 steps were reached). Note that this is slightly different from the learning process in that when you select an action during this testing process you should always choose the action with the highest Q value.

Experiments

Conduct several experiments to see how well your state representation works. Train your representation several times for a large number of games (100000) periodically stopping (after each 1000 games) to evaluate how good the solution is so far. Graph your results and discuss how your representation works.

What to Turn In

Submit a copy of your code. (12 points for well commented code)

Next you should write up a short report (at least one page, no more than three) discussing your design decisions in implementing the Q learning algorithm. (12 points for the report/documentation)

Problem 2: Model-based learning

Q-learning is a model-free reinforcement learning method because it does not learn the transition probabilities and immediate rewards directly. Implement a model-based reinforcement learning method that uses maximum likelihood estimates of the parameters (e.g. uses frequencies to estimate the transformation probabilities). Experimentally compare your method against Q-learning in terms of a) its rate of convergence and b) its final solution quality.

(16 points for the model-based code and comparison results)