15-889 Fall 2001 – Homework 3
Planning, Execution, and Learning

Due: Wednesday, November 21, 2001

Experimenting with Q-learning
(Thanks to Patrick Riley)

In this assignment, you will be experimenting with Q learning. We provide a library that simulates a world. You will write an agent that learns the optimal policy to act in the simulated world. The simulated world is achieved by a provided library in C. Therefore, you must do this assignment in C/C++, or in some language that can link in libraries written in C. All the files available are in `/afs/cs/user/reids/www/planning/hmw3/`

In the assignment, you will be able to implement and experiment with three different exploration strategies for Q-learning, which balance differently between exploration and exploitation:

Max-First This strategy is pure and extreme exploitation: the action with the highest $\widehat{Q}$ value is always chosen.

Random This strategy ignores the $\widehat{Q}$ values and picks randomly among the available action choices at every state.

Weighted Exploration/Exploitation This strategy allows to balance exploration and exploitation. The basic idea is to assign a probability to each action based on its current $\widehat{Q}$ value. How heavily the selection is weighted towards actions with high $\widehat{Q}$ values depends on a parameter k which your program will take as an argument.

Specifically, the formula for the probability of action a_i in state s is:

$$P(a_i|s) = \frac{k^{\widehat{Q}(s,a_i)}}{\sum_j k^{\widehat{Q}(s,a_j)}} \tag{1}$$

Notice that when $k = 1$, this is the random exploration strategy, and for very high values of k, this is like the max-first strategy.

In all cases, you should come up with a consistent methodology for breaking ties (this is usually particularly important at the beginning when most Q values are 0). We recommend that you pick randomly between ties, or choose an action which you have not tried before. Preferably, you would use both these strategies.

Updating Q Let γ be .9 for this assignment.

Please recall that, for deterministic worlds, the update of $\widehat{Q}$ is:

$$\widehat{Q}(s,a) \leftarrow r(s,a) + \gamma \max_{a'} \widehat{Q}(s',a') \tag{2}$$

And for non-deterministic worlds, the update rule is:

$$\widehat{Q}_n(s,a) \leftarrow (1-\alpha_n)\widehat{Q}_{n-1}(s,a) + \alpha_n[r(s,a) + \gamma \max_{a'} \widehat{Q}_{n-1}(s',a')] \tag{3}$$

where

$$\alpha_n = \frac{1}{1 + visits_n(s,a)} \tag{4}$$

Note that this rule is slower to converge than the deterministic rule.

The World Q-learning learns the world model and a policy. So the full description of the world is hidden from you, but you need to know a few things:

1. The number of states in the world is known (`MAX_STATES` to be specific).
2. The number of actions in the world is known (`MAX_ACTIONS` to be specific).
3. There are *absorbing* states, which means that, from some states, no action you can take will move you to a different state.
4. The world is fully observable, i.e., even in the non-deterministic version of the world, you still have perfect knowledge of your current state. The non-determinism is in the effects of the actions.
5. All actions in the world are reversible (except transitions into the absorbing states). This means that for every action taken, there is some action that will (usually) return you to the state you were just in. Therefore you need to be somewhat careful to avoid getting into cycles.

The world library The world that your agent acts in is pre-defined. The following functions are prototyped in `world.h`. In order to use them, you must link in the world library (Note that `world.h` also defines two types `State` and `Reward`)

- `void InitializeWorld(int mode, float chance_of_mismove)`

  This function must be called before any other world functions. The `mode` argument gives whether the world is to be deterministic or non-deterministic. `WM_DET` gives deterministic and `WM_NONDET` gives a non-deterministic world.

  The `chance_of_mismove` argument specifies the probability that the next state when applying the action is not the 'usual' one (ie the one that would occur in a deterministic world.

- `State GetCurrentState()`

  Gets the current state of the world.

- `Reward ApplyAction(int act)`

  Returns the reward by using action `act` in the current state. Returns `INVALID_REWARD` if the action is invalid. Note that in a non-deterministic world, this function could return different values when called with the same arguments.

- `int GetNumActionsOfState(State s)`

  Returns the number of actions available in state `s`. Actions are numbers that are 0 based, so if this function returns 2, then the valid actions are 0 and 1. Returns -1 on error.

- `bool Reset()`

  This function can *only* be called from an absorbing state. It resets the current state of the world to `INITIAL_STATE`.

The constants `MAX_ACTIONS`, `MAX_STATES`, and `INITIAL_STATE` are also defined in `world.h`, with the obvious meanings.

The Assignment You need to implement deterministic and non-deterministic Q learning, with all three exploration strategies discussed above. You should be able to implement it all in one program called `qlearn` and take command line arguments to specify what mode to be in.

Your program should be able to run the agent through multiple "epochs" of training. An epoch consists of putting the agent in the initial state and letting it move until it is stuck in an absorbing state.

Arguments Your program should accept the following command line arguments:

- `-m`, `-r`, or `-w` to indicate max-first, random, or weighted exploration strategy.
- `-d` or `-n` for deterministic or non-deterministic world
- `-k <val>` The k value for the weighted exploration strategy
- `-e <num>` to specify the number of epochs
- `-p <prob>` to specify the probability that (in a non-deterministic world) your action does not make it's 'usual' state transition

You should provide defaults in case these arguments are not specified.

Output You program should output the following to standard out, in the following order (all floating point values should be printed to 1 decimal place):

- The V value of every state (ie the max Q(s,a) over all actions). This should take this form: `state: <state> V: <value>` where the items in $<>$ represent the values.
- The average reward per epoch. This should be of the form

  `Average reward per epoch: <val>`

- The average number of steps per epoch. This should be of the form

  `Average steps per epoch: <val>`

- The average reward per step. This should be of the form

  `Average reward per step: <val>`

This output is designed to help you answer the questions below. You should not turn in all of this output for every test run you are asked to do.

Questions Once you have the basic program working, do the following experiments. If you feel a couple of lines of output from your program help justify your answer, please feel free to include them in your answer, otherwise do not provide output of the program.

For the deterministic case:

1. With the max-first exploration policy try various number of epochs between 1 and 15. How does increasing the epochs affect the learned Q values (in a very general way)? How many epochs does it take before you have a non zero Q value for the initial state?

2. Experiment with the random exploration strategy. Try numbers of epochs 10, 50, and 100. How much of the Q table is non-zero compared to the max first case? Which strategy gives better rewards? Can you explain why?

3. Now try the weighted exploration strategy with different values of k. At 1, it should be the same as the random strategy. What happens to the average reward as you increase k? What happens to the number of steps per epoch? What is the tradeoff increasing k (in other words, something get better when you increase k, but something else gets worse; what are those things?)

4. Now try to draw the state diagram for this world. The world should be simple to draw if you find the right arrangement of the states (that is, if the states are arranged correctly), then there is a simple geometric rule for which states are connected to which. You do not need to specify what action makes the transitions, just indicate which states have transitions between them. A huge list of pairs is *not* an acceptable answer here.

   Hopefully after doing this you will appreciate the work that your agent did *not* have to do (since you do not need to have or learn a world model for Q learning).

For the non-deterministic case:

1. Try all three exploration strategies with 5, 10, 50, 100 epochs. For each number of epochs, describe the relation of the average reward per step for each strategy. Since the world is non-deterministic, you will need to try each several times (5-10 times) to get a feel for the average value. If you can't find a consistent relationship between the strategies, that's okay! Just say that you can't find one. Note: Use .1 for p, and 2 for k.

2. Experiment with different values for the probability that a move takes you to the wrong state from .0 to .5. Use a fairly large number of epochs, such as 500. Is there a significant difference? Explain any differences (or the lack thereof).