CMPUT 366—Intelligent Systems

Assignment 4

Fall 2004
Department of Computing Science
University of Alberta

Due: *Wednesday, November 24 at 23:59:59 local time*
Worth: 10% of final grade
(8 questions worth 1% each and a 9th question worth 2%.)
Instructor: Dale Schuurmans, Ath409, x2-4806, dale@cs.ualberta.ca

---

Note: Submit eight Matlab functions in files called: pol2valR.m, optpolR.m, optvalR.m, val2polR.m, pol2valA.m, optpolA.m, optvalA.m, val2polA.m, and the answer to a question in a ninth file called q9.pdf.

Optimal behavior in a cat and mouse game

In this assignment you will implement simple methods for computing optimal behavior policies for a mouse that is trying to avoid a cat while simultaneously trying to eat cheese in a simple grid world. The first step is to go to the course webpage and get the tar file a4files.tar.gz. Gunzipping and untaring the file will create a subdirectory "a4files" which contains a definition of the environment (in environ.mat) and several Matlab M-files. In Matlab, type "`load environ.mat`." This will load the *global* variables `Adversary` (256×1), `Gamma` (1×1), `Possibs` ($256 \times 256 \times 5$), `Probs` ($256 \times 256 \times 5$), `Reward` (256×1). The included M-files are actdecode.m, actencode.m, animatepolicyA.m, animatepolicyR.m, applyA.m, applyR.m, enum.m, environ.mat, showaction.m, showstate.m, statedecode.m, and statencode.m—some of which are self explanatory, and some of which are explained below. To try things out, type `policy=ceil(rand(256,1)*5)` (which creates a random policy), `steps=20`, and then `animatepolicyR(policy,steps)` (or `animatepolicyA(policy,steps,0.5)`). When you do this you will see a little 4×4 grid world where a mouse (represented by a '3') is running around trying to eat cheese (represented by '1's at the corners) while avoiding the cat ('-8'). You are going to write programs that compute optimal behavior policies for two types of environments: environment `R` where the cat is unfortunately blind and just wanders around randomly, and environment `A` where the cat cannot only see, but in fact knows what the mouse's optimal next move is at every state (yikes!).

Representation

In this assignment, the world is a 4×4 grid of positions.

	0	1	2	3
0				
1				
2				
3				

Cheese is sitting at the corners $(0,0)$ and $(3,3)$. The two creatures in this world are (you) a mouse and (your opponent) a cat. Your goal is to eat cheese while avoiding the cat. In this world, a state can be described by four numbers (m_i, m_j, c_i, c_j) which give the i, j coordinates of the mouse (m_i, m_j) and the cat (c_i, c_j). Thus, there are a total of $4^4 = 256$ states. Both the cat and the mouse can execute one of five actions described by a pair of position differences:

(0, -1)	move left
(-1, 0)	move up
(0, 0)	stay
(1, 0)	move down
(0, 1)	move right

To make the implementation easier, the state vectors and action vectors will be recoded into state numbers (1 to 256) and action numbers (1 to 5) respectively. The Matlab M-files statedecode.m, statencode.m, actdecode.m, and actencode.m convert between the numerical and vector-based representations in an obvious way.

The immediate reward function for the mouse in this environment (given in the global variable `Reward`) is

$$\texttt{Reward}(s) = \begin{cases} 0 & \text{if the mouse is not on the cheese and the cat is not on the mouse} \\ 1 & \text{if the mouse is on the cheese and the cat is not on the mouse} \\ -3 & \text{if the mouse is not on the cheese but the cat is on the mouse} \\ -2 & \text{if the mouse is on the cheese but the cat is also on the mouse} \end{cases}$$

Of course, as shown in class, optimizing immediate reward is not usually sufficient for choosing actions that also affect the future states of the environment. Therefore, we really want to compute optimal behavior policies $\pi : S \rightarrow A$ that maximize a measure of the long term future reward the mouse obtains at every state. For this assignment we will specifically maximize the *expected future discounted reward* (as defined in class). The discount factor is stored in the global variable `Gamma` (with a default value of 0.95).

The goal will be to implement two different algorithms for computing optimal behavior policies, in two different types of environments. The first environment will be random, where nature (the cat) is oblivious to the mouse. The second environment will be adversarial, where the cat can actually anticipate the mouse's best moves. Interestingly, we will attempt to apply the same optimization algorithms for both types of maximization problems. However, the outcomes (as you can guess) wind up being very different.

Cat Model "R": Blind, oblivious, and random

In the first model the cat takes random moves at each time step. This will be called the (random) R-environment. The behavior for the cat in this environment is defined in the global variable `Probs` and in the M-file applyR.m. `Probs` contains the conditional probabilities for each next state of the environment, given the current state and the mouse's action. That is, `Probs(`sg`,`sn`,`a`)` $= \mathrm{P}(sn|sg, a)$, where sg is the given state number, sn is a next state number, and a is an action number. The function applyR(a,sg) picks a random next state given the current state sg and action a, according to the distribution in `Probs`.

For the routines in this first part of the assignment you will need to use the global environment variables `Gamma`, `Probs`, and `Reward`. Do not pass these as arguments to your functions, but rather include the definition "`global Gamma Probs Reward`" at the start of each function.

Question 1 (Evaluating a policy in R—1%)

Write a Matlab function "`pol2valR`" that takes one argument, `policy`, and returns a 256×1 matrix representing the *value function* for `policy` in the R-environment. To do this, note that the value function V for a policy $\pi : S \to A$ is defined to be

$$V_\pi(s) = R(s) + \gamma \sum_{s'} \mathrm{P}(s'|s, \pi(s))\; V_\pi(s')$$

for all states s, where γ is the discount factor (`Gamma`). One way to compute V given π is to solve a linear system of equations (with 256 unknowns and 256 equations). Another way to compute V is with an iterative procedure that starts with a random V and then iterates the equation

$$V_\pi^{new}(s) = R(s) + \gamma \sum_{s'} \mathrm{P}(s'|s, \pi(s))\; V_\pi^{old}(s')$$

The iteration stops when V^{new} and V^{old} are sufficiently close (say differing by less than `1e-10` on every state). You can implement either approach for this part. (It turns out that you have to use the second (iterative) approach later when computing the same thing for the A-environment.)

Question 2 (Optimizing a policy in R—1%)

Write a Matlab function "`optpolR`" that takes no arguments and returns a 256×1 matrix representing the optimal policy for the R-environment. Compute the optimal policy using *policy iteration*. That is, start with a random policy π, and compute its value function V. Given V, go through each state and update π as follows

$$\begin{aligned} \pi^{new}(s) &= \arg\max_a \;\; R(s) + \gamma \sum_{s'} \mathrm{P}(s'|s,a)\; V_{\pi^{old}}(s') \\ &= \arg\max_a \;\; \sum_{s'} \mathrm{P}(s'|s,a)\; V_{\pi^{old}}(s') \end{aligned}$$

If $\pi^{new} = \pi^{old}$, then halt and return it. Otherwise, re-compute the value function V for π^{new} and repeat.

Even though policy iteration seems like a good approach, it turns out that it is often computationally more efficient to instead compute the optimal value function first, and then solve for the optimal policy given this function. You will implement this alternative approach next (and compare the two).

Question 3 (Optimizing the value function in R—1%)

Write a Matlab function "`optvalR`" that takes no arguments and returns a 256×1 matrix representing the value function of the optimal policy for the R-environment. Compute the value function for the optimal policy using *value iteration*. That is, start with an arbitrary value function V, say $V = \frac{1}{1-\gamma} \max_s \texttt{Reward}(s)$, and update it iteratively as follows

$$V^{new}(s) \;\; = \;\; R(s) + \gamma \max_a \;\; \sum_{s'} \mathrm{P}(s'|s,a)\; V^{old}(s')$$

The iteration stops when V^{new} and V^{old} are sufficiently close (say differing by less than `1e-10` on every state).

Question 4 (Converting a value function to a policy in R—1%)

Write a Matlab function "`val2polR`" that takes one argument, `values`, and returns a 256×1 matrix representing the optimal policy for the given value function `values` in the R-environment. Given a value function V it is trivial to recover the optimal policy π for it by using

$$\begin{aligned} \pi(s) &= \arg\max_a \;\; R(s) + \gamma \sum_{s'} \mathrm{P}(s'|s,a)\; V(s') \\ &= \arg\max_a \;\; \sum_{s'} \mathrm{P}(s'|s,a)\; V(s') \end{aligned}$$

Cat Model "A": Alert, adversarial, and clairvoyant!

In the second model the cat will be able to take the optimal move (for the cat!) given whatever the mouse decides to do. Interestingly, the mouse can still adopt a sensible behavior strategy (namely, stay away from the cat!). However, to make things a bit more fair, we will add a parameter `rho` which specifies that the cat will sometimes take a random move with probability `rho`. That is, the cat is normally adversarial, but sometimes it will behave randomly (to make things more sporting for the mouse). This will be called the (adversarial) A-environment. The behavior for the cat in this environment is defined in the global variable `Possibs`, in the global variable `Probs`, in the parameter `rho`, and in the M-file applyA.m. `Possibs` indicates the possible next states in the environment, given the current state and the mouse's action. That is,

$$\texttt{Possibs}(sg, sn, a) = \begin{cases} 0 & \text{if state } sn \text{ is not possible by taking action } a \text{ in state } sg \\ 1 & \text{otherwise} \end{cases}$$

`Probs` has the same definition as before. The function applyA(a,sg,ρ) either picks the cat's best next state (that is, the mouse's worst next state) with probability $1 - \rho$, or makes a random move with probability ρ. When an adversarial move taken by the cat, given the current state sg and action a, it is determined according to the possibilities described in `Possibs`.

For the routines in this second part of the assignment you will need to use the global environment variables `Gamma`, `Possibs`, `Probs`, `Reward`, and `Adversary`. Do not pass these as arguments to your functions, but rather include the definition "`global Gamma Possibs Probs Reward Adversary`" at the start of each function.

Question 5 (Evaluating a policy in A—1%)

Write a Matlab function "`pol2valA`" that takes two arguments, `policy` and `rho`, and returns a 256×1 matrix representing the *value function* for `policy` in the A-environment, assuming the cat behaves randomly with probability `rho`. To do this, note that the value function V for a policy $\pi : S \to A$ in the A-environment is defined to be

$$V_\pi(s) \quad = \quad R(s) + \gamma(1-\rho)\left[\min_{s' \in S'(s,\pi(s))} V_\pi(s')\right] + \gamma\,\rho\,\left[\sum_{s'} \mathrm{P}(s'|s,\pi(s))\, V_\pi(s')\right]$$

for all states s, where γ is the discount factor (`Gamma`), ρ is the probability of a random cat move (`rho`), $S'(s,a)$ is the set of next states s' that are possible given current state s and action a (represented by `Possibs`), and $\mathrm{P}(s'|s,a)$ is the conditional probability of s' given s and a (represented by `Probs`). A way to attempt to compute V in the adversarial model is to use the same iterative procedure in Question 1. That is, start with a random V and then iterate the equation

$$V_\pi^{new}(s) \quad = \quad R(s) + \gamma(1-\rho)\left[\min_{s' \in S'(s,\pi(s))} V_\pi^{old}(s')\right] + \gamma\,\rho\,\left[\sum_{s'} \mathrm{P}(s'|s,\pi(s))\, V_\pi^{old}(s')\right]$$

The iteration stops when V^{new} and V^{old} are sufficiently close (say differing by less than `1e-10` on every state).

Question 6 (Optimizing a policy in A—1%)

Write a Matlab function "`optpolA`" that takes one argument, `rho`, and returns a 256×1 matrix representing the optimal policy for the A-environment. Compute the optimal policy using *policy iteration*. That is, start with a random policy π, and compute its value function V. Given V, go through each state and update π as follows

$$\begin{aligned}\pi^{new}(s) &= \arg\max_a \left\{ R(s) + \gamma(1-\rho)\left[\min_{s' \in S'(s,a)} V_{\pi^{old}}(s')\right] + \gamma\,\rho\,\left[\sum_{s'} \mathrm{P}(s'|s,a)\; V_{\pi^{old}}(s')\right]\right\} \\ &= \arg\max_a \left\{ (1-\rho)\left[\min_{s' \in S'(s,a)} V_{\pi^{old}}(s')\right] + \;\rho\,\left[\sum_{s'} \mathrm{P}(s'|s,a)\; V_{\pi^{old}}(s')\right]\right\}\end{aligned}$$

If $\pi^{new} = \pi^{old}$, then halt and return it. Otherwise, re-compute the value function V for π^{new} and repeat.

As in the previous R model, it is often computationally advantageous to compute the optimal value function first, and then solve for the optimal policy given this function. You will implement this alternative approach next (and compare the two).

Question 7 (Optimizing the value function in A—1%)

Write a Matlab function "`optvalA`" that takes one argument, `rho`, and returns a 256×1 matrix representing the value function of the optimal policy for the A-environment. Compute the value function for the optimal policy using *value iteration*. That is, start with an arbitrary value function V, say $V = \frac{1}{1-\gamma} \max_s \texttt{Reward}(s)$, and update it iteratively as follows

$$V^{new}(s) = R(s) + \gamma\,\max_a \left\{ (1-\rho)\left[\min_{s' \in S'(s,a)} V^{old}(s')\right] + \rho\,\left[\sum_{s'} \mathrm{P}(s'|s,a)\; V^{old}(s')\right]\right\}$$

The iteration stops when V^{new} and V^{old} are sufficiently close (say differing by less than `1e-10` on every state).

Question 8 (Converting a value function to a policy in A—1%)

Write a Matlab function "`val2polA`" that takes two arguments, `values` and `rho`, and returns a 256×1 matrix representing the optimal policy for the given value function `values` in the A-environment. Given a value function V it is trivial to recover the optimal policy π for it by using

$$\begin{aligned}\pi(s) &= \arg\max_a \left\{ R(s) + \gamma(1-\rho)\left[\min_{s' \in S'(s,a)} V(s')\right] + \gamma\,\rho\,\left[\sum_{s'} \mathrm{P}(s'|s,a)\; V(s')\right]\right\} \\ &= \arg\max_a \left\{ (1-\rho)\left[\min_{s' \in S'(s,a)} V(s')\right] + \rho\,\left[\sum_{s'} \mathrm{P}(s'|s,a)\; V(s')\right]\right\}\end{aligned}$$

Question 9 (2%)

Note that the A-environment can be turned into the R-environment by setting the parameter $\rho = 1$.

Solve for the optimal ρ-mouse in three different environments, where ρ is set to 0.0, 0.5, 1.0 respectively. This will give you 3 different mouse policies. For each mouse policy, calculate its value function in each of the 3 different ρ-environments using the procedure you implemented for Question 5. Note that there are 3 different mouse policies and 3 different environments, therefore there are 9 different mouse/environment combinations. Each mouse/environment combination yields a different value function. For each of the 9 value functions calculate the *mean* value (i.e. just the average of the values in V). Give a 3×3 table which shows the mean value obtained by each mouse policy in each environment.

Which mouse controller performs the best in each environment?

Next, run the R-mouse ($\rho = 1$) against the M-mouse ($\rho = 0.5$) and the A-mouse ($\rho = 0$) in the random (R) environment ($\rho = 1$) and then in the adversarial (A) environment $\rho = 0$. (That is, run three different mouse controllers in two different environments.) What is the difference in behavior between the three policies in the R-environment? In the A-environment? (1 sentence each)

Finally, if you didn't know which environment you would be faced with, which of the mouse controllers would you rather use? Why? (But explain what the drawbacks would be.)