Section 4: MDPs

CS 182 - Artificial Intelligence

Recall that a **Markov Decision Process** is defined by

- S: the set of possible states
- A: the set of possible actions, which may vary from state to state
- $T(s, a, s') \to \mathbb{R}$: the transition function is the probability that a from s leads to s'
- $R(s, a, s') \to \mathbb{R}$: the reward function. This could also be listed as $R(s')$ or $R(s, a)$
- The starting state
- And, potentially, a terminal state

With MDPs, we want to find the optimal policy $\pi^* : S \to A$, which maximizes the expected utility.

Since we need to maximize the expected utility, we need that utility to be finite. We can acomplish this in one of three ways:

1. Choosing a finite horizon: By terminating the game after T steps the reward must be finite. This often generates nonstationary policies, as π depends on time remaining.
2. Discounting over an infinite horizon: Multiplying rewards by the discount rate $0 < \gamma < 1$ every time a new step is made will cause the solution to converge to a finite return and thus a finite solution, where a smaller discount rate (γ) leads to a shorter term focus.

$$U([r_0, \cdots, r_\infty] = \sum_{t=0}^{\infty} \gamma^t r_t$$

3. Have an absorbing state: This will guarantee that for every policy a terminal state will eventually be reached.

We define a **Q-state** $Q(s, a)$ as the expected utility of having taken action a from state s and acting optimally afterward:

$$Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

And the **Value** of a state $V(s)$ as the expected utility of starting in s and acting optimally. This can also be thought of as taking the best action a at Q-state $Q(s, a)$:

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')] = \max_a Q^*(s, a)$$

Value Iteration is a way to compute the optimal value for every state. It is done by first initializing the values for each state to 0:

$$V_0(S) = 0$$

Then the $k+1$ iteration will take the vector of $V_k(s)$ for all states s and make a one step (Bellman) update:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$

Unfortunately, this process is quite slow, as while the optimal action at each state (and thus the optimal policy) rarely changes, the optimal values can take many iterations to finally converge. Furthermore, each iteration is $O(S^2A)$ and thus the dimensionality and time complexity of the problem grows quite quickly with the size of the state or action space.

Policy Iteration attempts to overcome these issues by iteratively alternating between **Policy Evaluation** (updating the values) and **Policy Improvement** (updating the policy). This both reduces time complexity of the algorithm and allows it to exit earlier once the policy converges.

Step 1: Policy Evaluation: Update the values under a fixed policy π_k:

$$V_{k+1}^{\pi_k}(s) = \sum_{s'} T(s, \pi_k(s), s')[R(s, \pi_k(s), s') + \gamma V_k^{\pi_k}(s')] = Q_k(s, \pi_k(s))$$

Step 2: Policy Improvement: Extract a better policy based on these new values:

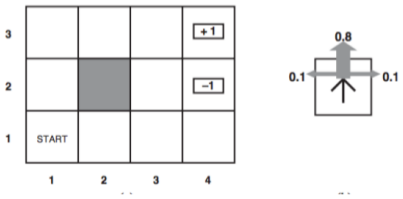
$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_{k+1}^{\pi_k}(s')] = \arg\max_a Q_{k+1}(s, a)$$

Note that Policy Evaluation is $O(S^2)$ operation and that both steps of Policy Iteration are easier to extract from the Q-values than the Values themselves.

Exercises

1. For the environment shown below, calculate which squares can be reached from $(1, 1)$ by the action sequence $[Up, Up, Right, Right]$ and with what probabilities. In this transition model, the probability of going in the intended direction is 0.8, but the agent travels in right angles to the intended direction with probability 0.1 each. A collision with a wall results in no movement.

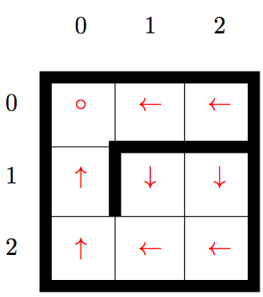
		Up	*Up*	*Right*	*Right*
(1,1)	1	.1	.02	.026	.0284
(1,2)		.8	.24	.258	.2178
(1,3)			.64	.088	.0346
(2,1)		.1	.09	.034	.0276
(2,3)				.512	.1728
(3,1)			.01	.073	.0346
(3,2)				.001	.0073
(3,3)					.4097
(4,1)				.008	.0656
(4,2)					.0016
(4,3)					



2. Pacman is stuck in a friendlier maze where he gets a reward of 1 every time he takes an action from state (0,0). This setup is a bit different from the one you have seen before: Pacman can get the reward multiple times; these rewards do not get "used up" like food pellets and there are no "living rewards." As usual, Pacman cannot move through walls and may take any of the following actions: go North (↑), South (↓), East (→), West (←), or stay in place (○). State (0,0) gives a total reward of 1 every time Pacman takes an action in that state regardless of the outcome, and all other states give no reward. Actions are deterministic: each action will take Pacman in the correct direction unless Pacman runs into a wall, in which case he will remain in the same square.

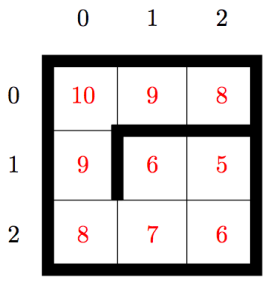
   a) Assume a finite horizon of $h = 10$ (so Pacman takes exactly 10 steps) and no discounting ($\gamma = 1$).

Fill in an optimal policy:



(available actions: ↑, ↓, →, ←, ○)

Fill in the value function:



   b) Fill in following Q-values (with 10 steps remaining) corresponding to the value function you specified above.
   The Q value of state-action (0,0), (East) is:

   $Q_{10}(S, A) = R(S, A) + V_9(S') = 1 + 8 = 9$

   The Q value of state-action (1,1), (East) is:

   $Q_{10}(S, A) = R(S, A) + V_9(S') = 0 + 4 = 4$

   c) Now assume an infinite horizon, with $\gamma = 0.9$. Find the value of state (0,0):

   Since the only reward is in state (0,0) if we start in state (0,0) we would choose to stay there forever. Therefore we will accumulate the following reward: $\sum_{i=1}^{\infty} 0.9^i * 1$. This is an infinite geometric series with a ratio < 1 and therefore the closed form solution is: $V = V_0/(1 - \gamma) = 1/(1 - 0.9) = 10$.

   d) Now let $\gamma = 1$. Also, now at every time step, after Pacman takes an action and collects his reward, a power outage could suddenly end the game with probability $\alpha = 0.1$. Find the value of state (0,0):

   Again we would like to stay in state (0,0) forever. However, there is a chance α that the game ends on each turn therefore we expect the game to go for $1/\alpha$ rounds where we get reward 1. $V = 1/\alpha = 10$.

   Another approach to this problem would calculate the value by doing an infinite geometric sum $1 + (1 - \alpha) + (1 - \alpha^2) + ... = 10$, essentially treating the probability of ending the game as a discount.

3. Consider an undiscounted MDP having three states, (1, 2, 3), with rewards $-1, -2, 0$, respectively. State 3 is a terminal state. In states 1 and 2 there are two possible actions: A and B. The transition model is as follows:

   - In state 1, action A moves the agent to state 2 with probability 0.8 and makes the agent stay put with probability 0.2.
   - In state 2, action A moves the agent to state 1 with probability 0.8 and makes the agent stay put with probability 0.2.
   - In either state 1 or state 2, action B moves the agent to state 3 with probability 0.1 and makes the agent stay put with probability 0.9.

   a) Without running policy iteration, what can be determined about the optimal policy in states 1 and 2? Which actions minimize the reward lost while staying in states 1 and 2 while maximizing the probability of reaching the terminal state?

   Intuitively, the agent wants to get to state 3 as soon as possible, because it will pay a cost for each time step it spends in states 1 and 2. However, the only action that reaches state 3 (action b) succeeds with low probability, so the agent should minimize the cost it incurs while trying to reach the terminal state. This suggests that the agent should definitely try action b in state 1; in state 2, it might be better to try action a to get to state 1 (which is the better place to wait for admission to state 3), rather than aiming directly for state 3. The decision in state 2 involves a numerical tradeoff.

   b) Apply policy iteration, showing each step in full, to determine the optimal policy and the values of states 1 and 2. Assume that the initial policy has action B in both states and initial values of 0 for all states and $\gamma = 1$.

   Initialization
   Values are V(State 1) $= 0$, V(State 2) $= 0$, V(State 3) $= 0$.
   Policy is π(State 1) $= B$, π(State 2) $= B$.

   Iteration 1
   Calculate new values by following current policy:
   V(State 1) $= 0.9(-1 + \gamma * 0) + 0.1(0 + 0) = -0.9$
   V(State 2) $= 0.9(-2 + \gamma * 0) + 0.1(0 + 0) = -1.8$

   Update the policy by doing a one-step lookahead, with the values that were just computed:
   Q(State 1, Action A) $= 0.2(-1 + \gamma * (-0.9)) + 0.8(-2 + \gamma * (-1.8)) = -3.42$
   Q(State 1, Action B) $= 0.9(-1 + \gamma * (-0.9)) + 0.1(0 + \gamma * 0) = -1.71$
   Q(State 2, Action A) $= 0.2(-2 + \gamma * (-1.8)) + 0.8(-1 + \gamma * (-0.9)) = -2.28$
   Q(State 2, Action B) $= 0.9(-2 + \gamma * (-1.8)) + 0.1(0 + 0) = -3.42$

   Policy update for this iteration: π(State 1) $= B$, π(State 2) $= A$.

   Iteration 2
   The value updates have already been calculated during the policy update above:
   V(State 1) $= -1.71$

V(State 2) $= -2.28$

Update the policy by doing a one-step lookahead, with the values that were just computed:
Q(State 1, Action A) $= 0.2(-1 + \gamma * (-1.71)) + 0.8(-2 + \gamma * (-2.28)) = -3.966$
Q(State 1, Action B) $= 0.9(-1 + \gamma * (-1.71)) + 0.1(0 + \gamma * 0) = -2.439$
Q(State 2, Action A) $= 0.2(-2 + \gamma * (-2.28)) + 0.8(-1 + \gamma * (-1.71)) = -3.024$
Q(State 2, Action B) $= 0.9(-2 + \gamma * (-2.28)) + 0.1(0 + 0) = -3.852$

Policy update for this iteration: π(State 1) $= B$, π(State 2) $= A$.

Iteration 3
Algorithm terminates because policy has converged (it has not changed between iterations). Note that the values have not converged, and in this particular problem, will continue to become more negative as the algorithm continues.