CS 188
Spring 2023

Midterm Review MDPs Solutions

Markov Decision Processes

A Markov Decision Process is defined by several properties:

- A set of states S
- A set of actions A.
- A start state.
- Possibly one or more terminal states.
- Possibly a **discount factor** γ.
- A **transition function** $T(s, a, s')$.
- A **reward function** $R(s, a, s')$.

The Bellman Equation

- $V^*(s)$ – the optimal value of s is the expected value of the utility an optimally-behaving agent that starts in s will receive, over the rest of the agent's lifetime.
- $Q^*(s, a)$ - the optimal value of (s, a) is the expected value of the utility an agent receives after starting in s, taking a, and acting optimally henceforth.

Using these two new quantities and the other MDP quantities discussed earlier, the Bellman equation is defined as follows:

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

We can also define he equation for the optimal value of a q-state (more commonly known as an optimal **q-value**):

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

which allows us to reexpress the Bellman equation as

$$V^*(s) = \max_a Q^*(s, a).$$

Value Iteration

The **time-limited value** for a state s with a time-limit of k timesteps is denoted $V_k(s)$, and represents the maximum expected utility attainable from s given that the Markov decision process under consideration terminates in k timesteps. Equivalently, this is what a depth-k expectimax run on the search tree for a MDP returns.

Value iteration is a **dynamic programming algorithm** that uses an iteratively longer time limit to compute time-limited values until convergence (that is, until the V values are the same for each state as they were in the past iteration: $\forall s, V_{k+1}(s) = V_k(s)$). It operates as follows:

1. $\forall s \in S$, initialize $V_0(s) = 0$. This should be intuitive, since setting a time limit of 0 timesteps means no actions can be taken before termination, and so no rewards can be acquired.
2. Repeat the following update rule until convergence:
$$\forall s \in S,\ V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$
At iteration k of value iteration, we use the time-limited values for with limit k for each state to generate the time-limited values with limit $(k+1)$. In essence, we use computed solutions to subproblems (all the $V_k(s)$) to iteratively build up solutions to larger subproblems (all the $V_{k+1}(s)$); this is what makes value iteration a dynamic programming algorithm.

Policy Extraction

Recall that our ultimate goal in solving a MDP is to determine an optimal policy. This can be done once all optimal values for states are determined using **policy extraction**. The intuition is simple: if you're in a state s, you should take the action a which yields the maximum expected utility. Not surprisingly, a is the action which takes us to the q-state with maximum q-value, allowing for a formal definition of the optimal policy:
$$\forall s \in S,\ \pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

Policy Iteration

If all we want is to determine the optimal policy for the MDP value iteration tends to do a lot of overcomputation since the policy as computed by policy extraction generally converges significantly faster than the values themselves. This motivates **policy iteration**, an algorithm that maintains the optimality of value iteration while providing significant performance gains. It operates as follows:

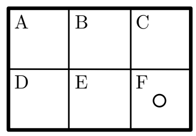
1. Define an *initial policy*. This can be arbitrary, but policy iteration will converge faster the closer the initial policy is to the eventual optimal policy.
2. Repeat the following until convergence:
   - **Policy evaluation:** For a policy π, policy evaluation means computing $V^\pi(s)$ for all states s, where $V^\pi(s)$ is expected utility of starting in state s when following π:
   $$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$$
   Define the policy at iteration i of policy iteration as π_i. Since we are fixing a single action for each state, we no longer need the max operator which effectively leaves us with a system of $|S|$ equations generated by the above rule. Each $V^{\pi_i}(s)$ can then be computed by simply solving this system.
   - **Policy improvement:** Policy improvement uses policy extraction on the values of states generated by policy evaluation to generate this new and improved policy:
   $$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}}\, Q^{\pi_i}(s,a) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$$
   If $\pi_{i+1} = \pi_i$, the algorithm has converged, and we can conclude that $\pi_{i+1} = \pi_i = \pi^*$.

Q1. MDP

Pacman is using MDPs to maximize his expected utility. In each environment:

- Pacman has the standard actions {North, East, South, West} unless blocked by an outer wall
- There is a reward of 1 point when eating the dot (for example, in the grid below, $R(C, South, F) = 1$)
- The game ends when the dot is eaten

(a) Consider a the following grid where there is a single food pellet in the bottom right corner (F). The **discount** factor is 0.5. There is no living reward. The states are simply the grid locations.



(i) What is the optimal policy for each state?

State	$\pi(state)$
A	East or South
B	East or South
C	South
D	East
E	East

(ii) What is the optimal value for the state of being in the upper left corner (A)? Reminder: the discount factor is 0.5.

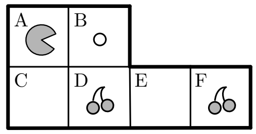
$V^*(A) =$ 0.25

k	V(A)	V(B)	V(C)	V(D)	V(E)	V(F)
0	0	0	0	0	0	0
1	0	0	1	0	1	0
2	0	0.5	1	0.5	1	0
3	0.25	0.5	1	0.5	1	0
4	0.25	0.5	1	0.5	1	0

(iii) Using value iteration with the value of all states equal to zero at k=0, for which iteration k will $V_k(A) = V^*(A)$?

$k =$ 3 (see above)

(b) Consider a new Pacman level that begins with cherries in locations D and F. Landing on a grid position with cherries is worth 5 points and then the cherries at that position disappear. There is still one dot, worth 1 point. The game still only ends when the dot is eaten.



(i) With no discount ($\gamma = 1$) and a living reward of -1, what is the optimal policy for the states in this level's state space?

State	$\pi(state)$
A, D_{Cherry}=true, F_{Cherry}=true	South
A, D_{Cherry}=true, F_{Cherry}=false	South
A, D_{Cherry}=false, F_{Cherry}=true	East
A, D_{Cherry}=false, F_{Cherry}=false	East
C, D_{Cherry}=true, F_{Cherry}=true	East
C, D_{Cherry}=true, F_{Cherry}=false	East
C, D_{Cherry}=false, F_{Cherry}=true	East
C, D_{Cherry}=false, F_{Cherry}=false	North/East
D, D_{Cherry}=false, F_{Cherry}=true	East
D, D_{Cherry}=false, F_{Cherry}=false	North
E, D_{Cherry}=true, F_{Cherry}=true	East
E, D_{Cherry}=true, F_{Cherry}=false	West
E, D_{Cherry}=false, F_{Cherry}=true	East
E, D_{Cherry}=false, F_{Cherry}=false	West
F, D_{Cherry}=true, F_{Cherry}=false	West
F, D_{Cherry}=false, F_{Cherry}=false	West

(ii) With no discount ($\gamma = 1$), what is the range of living reward values such that Pacman eats exactly one cherry when starting at position A?

Valid range for the living reward is (-2.5,-1.25).

Let x equal the living reward.

The reward for eating zero cherries {A,B} is $x + 1$ (one step plus food).

The reward for eating exactly one cherry {A,C,D,B} is $3x + 6$ (three steps plus cherry plus food).

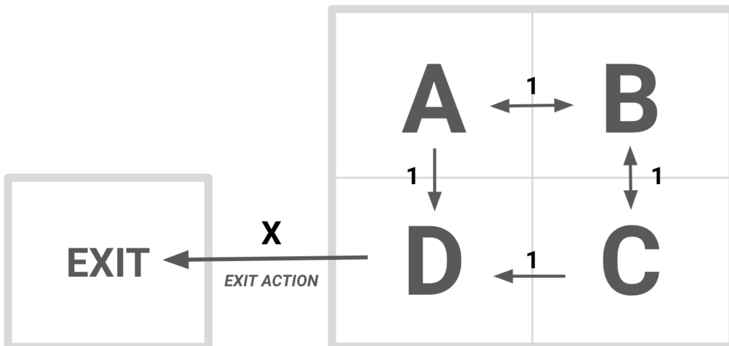
The reward for eating two cherries {A,C,D,E,F,E,D,B} is $7x + 11$ (seven steps plus two cherries plus food).

x must be greater than -2.5 to make eating at least one cherry worth it ($3x + 6 > x + 1$).

x must be less than -1.25 to eat less than one cherry ($3x + 6 > 7x + 11$).

Q2. Strange MDPs

In this MDP, the available actions at **state A, B, C** are *LEFT, RIGHT, UP,* and *DOWN* unless there is a wall in that direction. The only action at **state D** is the *EXIT ACTION* and gives the agent a **reward of** x. The **reward for non-exit actions is always 1**.



(a) Let all actions be deterministic. Assume $\gamma = \frac{1}{2}$. Express the following in terms of x.

$V^*(D) = x$

$V^*(C) = max(1 + 0.5x, 2)$

$V^*(A) = max(1 + 0.5x, 2)$

$V^*(B) = max(1 + 0.5(1 + 0.5x), 2)$

The 2 comes from the utility being an infinite geometric sum of discounted reward $= \frac{1}{(1-\frac{1}{2})} = 2$

(b) Let any non-exit action be successful with probability $= \frac{1}{2}$. Otherwise, the agent stays in the same state with reward $= 0$. The *EXIT ACTION* from the **state D** is still deterministic and will always succeed. Assume that $\gamma = \frac{1}{2}$.

For which value of x does $Q^*(A, DOWN) = Q^*(A, RIGHT)$? Box your answer and justify/show your work.

$Q^*(A, DOWN) = Q^*(A, RIGHT)$ implies $V^*(A) = Q^*(A, DOWN) = Q^*(A, RIGHT)$

$$V^*(A) = Q^*(A, DOWN) = \frac{1}{2}(0 + \frac{1}{2}V^*(A)) + \frac{1}{2}(1 + \frac{1}{2}x) = \frac{1}{2} + \frac{1}{4}(V^*(A)) + \frac{1}{4}x \tag{1}$$

$$V^*(A) = \frac{2}{3} + \frac{1}{3}x \tag{2}$$

$$V^*(A) = Q^*(A, RIGHT) = \frac{1}{2}(0 + \frac{1}{2}V^*(A)) + \frac{1}{2}(1 + \frac{1}{2}V^*(B)) = \frac{1}{2} + \frac{1}{4}V^*(A) + \frac{1}{4}V^*(B) \tag{3}$$

$$V^*(A) = \frac{2}{3} + \frac{1}{3}V^*(B) \tag{4}$$

Because $Q^*(B, LEFT)$ and $Q^*(B, DOWN)$ are symmetric decisions, $V^*(B) = Q^*(B, LEFT)$.

$$V^*(B) = \frac{1}{2}(0 + \frac{1}{2}V^*(B)) + \frac{1}{2}(1 + \frac{1}{2}V^*(A)) = \frac{1}{2} + \frac{1}{4}V^*(B) + \frac{1}{4}V^*(A) \tag{5}$$

$$V^*(B) = \frac{2}{3} + \frac{1}{3}V^*(A) \tag{6}$$

Combining (2), (4), and (6) gives us: x = 1

(c) We now add one more layer of complexity. Turns out that the reward function is not guaranteed to give a particular reward when the agent takes an action. Every time an agent transitions from one state to another, once the agent reaches the new state s', a fair 6-sided dice is rolled. If the dices lands with value x, the agent receives the reward $R(s, a, s') + x$. The sides of dice have value $1, 2, 3, 4, 5$ and 6.

Write down the new bellman update equation for $V_{k+1}(s)$ in terms of $T(s, a, s')$, $R(s, a, s')$, $V_k(s')$, and γ.

$V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[\frac{1}{6}(\sum_{i=1}^{6} R(s, a, s') + i) + \gamma V_k(s')]$

$= \max_a \sum_{s'} T(s, a, s')(R(s, a, s') + 3.5 + \gamma V_k(s'))$