CS221 Fall 2018 Homework 4

SUNet ID: 05794739
Name: Luis Perez
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Problem 1

(a) We give the value for each iteration. We note that $V_{\text{opt}}^0(s) = 0$ to start out. We also note that since for $s_t \in \{-2, 2\}$ we are at a terminal state, we'll have $V_{\text{opt}}(s_t) = 0$ for all iterations.

(a) After iteration 0, we'll have:

$$\begin{aligned} V_{\text{opt}}^0(-1) &= 0 \\ V_{\text{opt}}^0(0) &= 0 \\ V_{\text{opt}}^0(1) &= 0 \end{aligned}$$

(b) After the first iteration, we'll have the following values:

$$\begin{aligned} V_{\text{opt}}^1(-1) &= \max_{a\in\{-1,1\}}\{0.8[20 + V_{\text{opt}}^0(-2)] + 0.2[-5 + V_{\text{opt}}^0(0)], 0.7[20 + V_{\text{opt}}^0(-2)] + 0.3[-5V_{\text{opt}}^0(0)]\} \\ &= 15 \\ V_{\text{opt}}^1(0) &= \max_{a\in\{-1,1\}}\{0.8[-5 + V_{\text{opt}}^0(-1)] + 0.2[-5 + V_{\text{opt}}^0(1)], 0.7[-5 + V_{\text{opt}}^0(-1)] + 0.3[-5 + V_{\text{opt}}^0(\\ &= -5 \\ V_{\text{opt}}^1(1) &= \max_{a\in\{-1,1\}}\{0.8[-5 + V_{\text{opt}}^0(0)] + 0.2[100 + V_{\text{opt}}^0(2)], 0.7[-5 + V_{\text{opt}}^0(0)] + 0.3[100 + V_{\text{opt}}^0(2)\} \\ &= 26.5 \end{aligned}$$

(c) Finally, after the second iteration, we'll have:

$$\begin{aligned} V_{\text{opt}}^2(-1) &= \max_{a\in\{-1,1\}}\{0.8[20 + V_{\text{opt}}^1(-2)] + 0.2[-5 + V_{\text{opt}}^1(0)], 0.7[20 + V_{\text{opt}}^1(-2)] + 0.3[-5 + V_{\text{opt}}^1(0) \\ &= 14 \\ V_{\text{opt}}^2(0) &= \max_{a\in\{-1,1\}}\{0.8[-5 + V_{\text{opt}}^1(-1)] + 0.2[-5 + V_{\text{opt}}^1(1)], 0.7[-5 + V_{\text{opt}}^1(-1)] + 0.3[-5 + V_{\text{opt}}^1(\\ &= 13.45 \\ V_{\text{opt}}^2(1) &= \max_{a\in\{-1,1\}}\{0.8[-5 + V_{\text{opt}}^1(0)] + 0.2[100 + V_{\text{opt}}^1(2)], 0.7[-5 + V_{\text{opt}}^1(0)] + 0.3[100 + V_{\text{opt}}^1(2)\} \\ &= 23 \end{aligned}$$

(b) We interpret this question as asking for the resulting optimal policy for non-terminal states after two iterations. In that case, we have:

$$\begin{aligned}
\pi^2_{\text{opt}}(-1) &= \arg\max_{a\in\{-1,1\}} \{0.8[20 + V^1_{\text{opt}}(-2)] + 0.2[-5 + V^1_{\text{opt}}(0)], 0.7[20 + V^1_{\text{opt}}(-2)] + 0.3[-5 + V^1_{\text{opt}}(0)] \\
&= -1 \\
\pi^2_{\text{opt}}(0) &= \arg\max_{a\in\{-1,1\}} \{0.8[-5 + V^1_{\text{opt}}(-1)] + 0.2[-5 + V^1_{\text{opt}}(1)], 0.7[-5 + V^1_{\text{opt}}(-1)] + 0.3[-5 + V^1_{\text{opt}}(1) \\
&= 1 \\
\pi^2_{\text{opt}}(1) &= \arg\max_{a\in\{-1,1\}} \{0.8[-5 + V^1_{\text{opt}}(0)] + 0.2[100 + V^1_{\text{opt}}(2)], 0.7[-5 + V^1_{\text{opt}}(0)] + 0.3[100 + V^1_{\text{opt}}(2)\} \\
&= 1
\end{aligned}$$

Problem 2

(a) It is not always the case that $V_1(s_{\text{start}}) \geq V_2(s_{\text{start}})$. For a counter-examples, see "submission.py".

(b) The algorithm is rather straight-forward in the case where we have an acyclic MDP.

- The first-step in the algorithm is to topologically sort the graph. It is well-known that for a DAG, a topological sorting is possible and can be computed by a modified version of DFS in linear time [1]
- Once we have this topological sorting of the states, we process each state in reverse-topological order and compute

$$V(s) = \max_{a\in A}\left\{\sum_{s'\in\text{Succ}(s,a)} T(s,a,s')[R(s,a,s') + V(s')]\right\}$$

directly for each such state.
- After this single pass over the states, we return the resulting value function.

We claim that the computed $V(s) = V_{\text{opt}}(s)$ (ie, in this single pass, we've computed the optimal value function). To undertand why, we must recall that a topological sorting is one such that for every edge transition (s, a, s'), from s to s', s comes before s' in the ordering. In our algorithm above, we processed these states in reverse order (ie, we calculate the value of s' before we compute the value of s). More formally, consider all terminal states (ie, states with no successors). These states are processed first by our algorithm, given us the base case:

$$V(s') = 0 = V_{\text{opt}}(s') \qquad \text{(for all terminal states } s'\text{)}$$

[1] `https://en.wikipedia.org/wiki/Topological_sorting#Depth-first_search`

Now let us assume $V(s') = V_{\text{opt}}(s')$ for all s' which our algorithm has already processed (ie, if our algorithm is processing states s, then the above holds true for all states s' which fall after s in the toplogical sort). Consider the processing of state s. For this state, our algorithm will compute:

$$\begin{aligned} V(s) &= \max_{a \in A} \left\{ \sum_{s' \in \text{Succ}(s,a)} T(s,a,s')[R(s,a,s') + V(s')] \right\} && \text{(definition of our algorithm)} \\ &= \max_{a \in A} \left\{ \sum_{s' \in \text{Succ}(s,a)} T(s,a,s')[R(s,a,s') + V_{\text{opt}}(s')] \right\} \\ & \text{(all } s' \text{ are descendants of } s\text{, and therefore, by the inductive hypothesis, we have } V(s') = V_{\text{opt}}(s')) \\ &= V_{\text{opt}}(s) && \text{(definition of } V_{\text{opt}}) \end{aligned}$$

(c) Following the hint, the solution to this problem is essentially given to us in lecture. The problem already provides States', Actions'(s), and γ'. As per Percy's lecture notes, we define the transition probabilities and rewards as follows:

$$T'(s,a,s') = \begin{cases} (1-\gamma) & s' = o \\ \gamma T(s,a,s') & \text{otherwise} \end{cases}$$

$$R'(s,a,s') = \begin{cases} 0 & s' = o \\ R(s,a,s') & \text{otherwise} \end{cases}$$

Informally, with probability $(1-\gamma)$ every state can now end in a terminal state with (receiving 0 reward). All other transitions probabilities are discounted by γ. We claim that $V_{\text{opt}}(s) = V'_{\text{opt}}$ for all $s \in$ States. We can prove this directly. First, let's recall that if $V_{\text{opt}}(s)$ exists, it is the unique solution to:

$$V_{\text{opt}}(s) = \max_{a \in \text{Actions}(s)} \left\{ \sum_{s' \in \text{Succ}(s,a)} T(s,a,s')[R(s,a,s') + V_{\text{opt}}(s'))] \right\}$$

Now let us consider a state $s \in \text{States}$. Then we have:

$$\begin{aligned}
V'_{\text{opt}}(s) &= \max_{a \in \text{Actions'}(s)} \left\{ \sum_{s' \in \text{Succ'}(s,a)} T'(s,a,s')[R(s,a,s') + V'_{\text{opt}}(s')] \right\} \text{ (definition of } V'_{\text{opt}}) \\
&= \max_{a \in \text{Actions}(s)} \left\{ T'(s,a,o)[R'(s,a,o) + V'_{\text{opt}}(o)] + \sum_{s' \in \text{Succ}(s,a)} T'(s,a,s')[R'(s,a,s') + V'_{\text{opt}}(s')] \right\} \\
&\quad (\text{Actions}'(s) = \text{Actions}(s) \text{ and } \text{Succ}'(s,a) = \{o\} \cup \text{Succ}(s,a) \text{ by contruction}) \\
&= \max_{a \in \text{Actions}(s)} \left\{ \sum_{s' \in \text{Succ}(s,a)} \gamma T(s,a,s')[R(s,a,s') + V'_{\text{opt}}(s')] \right\}. \\
&\quad (R'(s,a,o) + V'_{\text{opt}}(o) = 0 \text{ and definition of } R' \text{ and } T')
\end{aligned}$$

Note that the above equation is precisely the equation that $V_{\text{opt}}(s)$ solves. Therefore, we must have that $V_{\text{opt}}(s)$ and $V'_{\text{opt}}(s)$ are the same function.

Problem 3

(a) In "submission.py".

(b) In "submission.py".

Problem 4

(a) In "submission.py".

(b) The policy learned by Q-Learning matches exactly the policy learned by value iteration in the small MDP. However, in the large MDP, the policy learned by Q-Learning differs from the optimal policy significantly (in our experiment, for example, 752 our of 2705 states have non-optimal actions selected by our Q-Learning algorithm).

The reason for this is relatively straight-forward. Due to the large nature of the MDP, even after 30,000 trials, our ϵ-greedy learning policy will still leave a significant portion of the state-action space unexplored. This is further exacerbated by the fact that our feature extractor for this section is the identityFeatureExtractor, which treats each state-action pair completely independently. This means that for all of the un-explored state-action space, we actually have learned *nothing* and cannot generalize.

As such, for this much larger MDP, Q-learning performs poorly.

(c) In "submission.py".

(d) The sampled average reward for optimal policy from original MDP run on the modified MDP is 6.83776666667 (over 30000 simulations run with the optimal policy). If instead we train directly on the modified MDP, the sampled average reward for Q-Learning during training is 9.58873333333 (over 30000 training simulations). Using the optimal policy after these 30000 training simulations, the sampled avarage reward of this new optimal policy (without further training) is 12.0.

As we can see, the rewards are best when training directly on the new MDP, rather than using even the optimial policy from the original MDP. This makes sense, since taking an optimal policy from one MDP to another means that we'll be making poor decisions in particular states. For the blackjack example, for example, the original optimal policy is likely to 'Take' on 16 – however, this is certainly no longer the case is the threshold is lowered. It's much better, therefore, to just learn from the new MDP.