Homework 4: Peeking Blackjack

Brandon McKinzie

Problem 1: Value Iteration

In this problem, you will perform the value iteration updates manually on a very basic game just to solidify your intuitions about solving MDPs. The set of possible states in this game is {-2, -1, 0, 1, 2}. You start at state 0, and if you reach either -2 or 2, the game ends. At each state, you can take one of two actions: {-1, +1}.

If you're in state s and choose +1:

- *You have a 70% chance of reaching the state s+1.*
- *You have a 30% chance of reaching the state s-1.*

If you're in state s and choose -1:

- *You have a 20% chance of reaching the state s+1.*
- *You have an 80% chance of reaching the state s-1.*

If your action results in transitioning to state -2, then you receive a reward of 20. If your action results in transitioning to state 2, then your reward is 100. Otherwise, your reward is -5. Assume the discount factor γ is 1.

(a) *Give the value of $V_{opt}(s)$ for each state s after 0, 1, and 2 iterations of value iteration. Iteration 0 just initializes all the values of V to 0. Terminal states do not have any optimal policies and take on a value of 0.*

By definition $V_{opt}^{(0)}(s) \leftarrow 0$. Using this and the fact that $\gamma = 1$, we can compute

$$V_{opt}^{(1)}(s) = \max_{a\in\{+1,-1\}} \sum_{s'\in\{s+1,s-1\}} T(s,a,s')R(s,a,s') \tag{1}$$

$$= \max\{0.7R(s,1,s{+}1){+}0.3R(s,1,s{-}1),\ 0.2R(s,-1,s{+}1){+}0.8R(s,-1,s{-}1)\} \tag{2}$$

$$V_{opt}^{(1)}(0) = \max\{0.7(-5){+}0.3(-5),\ 0.2(-5){+}0.8(-5)\} = -5 \tag{3}$$

$$V_{opt}^{(1)}(-1) = \max\{0.7(-5){+}0.3(20),\ 0.2(-5){+}0.8(20)\} = 15 \tag{4}$$

$$V_{opt}^{(1)}(1) = \max\{0.7(100){+}0.3(-5),\ 0.2(100){+}0.8(-5)\} = 68.5 \tag{5}$$

$$V_{opt}^{(1)}(-2) = V_{opt}^{(1)}(2) = 0 \tag{6}$$

We then use these values to compute the next iteration.

$$
\begin{aligned}
V_{opt}^{(2)}(s) &= \max_{a\in\{+1,-1\}} \sum_{s'\in\{s+1,s-1\}} T(s,a,s')\left[R(s,a,s') + V_{opt}^{(1)}(s')\right] && (7)\\
V_{opt}^{(2)}(0) &= \max\left\{0.7(-5+68.5)+0.3(-5+15),\ 0.2(-5+68.5)+0.8(-5+15)\right\} && (8)\\
&= 47.45 && (9)\\
V_{opt}^{(2)}(-1) &= \max\left\{0.7(-5-5)+0.3(20), 0.2(-5-5)+0.8(20)\right\} && (10)\\
&= 14 && (11)\\
V_{opt}^{(2)}(1) &= \max\left\{0.7(100)+0.3(-5-5), 0.2(100)+0.8(-5-5)\right\} && (12)\\
&= 67 && (13)\\
V_{opt}^{(2)}(-2) &= V_{opt}^{(2)}(2) = 0 && (14)
\end{aligned}
$$

(b) *What is the resulting optimal policy π_{opt} for all non-terminal states?*

$$\pi_{opt}(0) = +1 \tag{15}$$
$$\pi_{opt}(-1) = -1 \tag{16}$$
$$\pi_{opt}(1) = +1 \tag{17}$$

Problem 2: Transforming MDPs

Let's implement value iteration to compute the optimal policy on an arbitrary MDP. Later, we'll create the specific MDP for Blackjack.

(a) I provided a counterexample in the code.

(b) *Suppose we have an acyclic MDP for which we want to find the optimal value at each node. We could run value iteration, which would require multiple iterations – but it would be nice to be more efficient for MDPs with this acyclic property. Briefly explain an algorithm that will allow us to compute V_{opt} for each node with only a single pass over all the triples.*

For acyclic graphs, we can work our way from terminal (end-state) nodes back to the start state node. This will allow us to only do a single pass over the (s, a, s') triples because for all of the end state nodes s_{end}, we'll have $V_{opt}(s_{end}) = 0$. All edges that originate from a chance node to an end state node will be computed next. Specifically, for each end-state node s_{end}, we'll compute

$$T(s, a, s_{end})[R(s, a, s_{end}) + \gamma \cancel{V_{opt}(s_{end})}] \qquad \forall (s, a) : T(s, a, s_{end}) > 0 \tag{18}$$

and continue working our way back until we reach the start state.

TODO: may want to elaborate more here.

(c) *Suppose we have an MDP with statesStates and a discount factor $\gamma < 1$, but we have an MDP solver that can only solve MDPs with discount factor of 1. How can we leverage the MDP solver to solve the original MDP?*

Let us define a new MDP with states $States' = States \cup \{o\}$, where o is a new state. Let's use the same actions ($Actions'(s) = Actions(s)$), but we need to keep the discount $\gamma' = 1$. Your job is to define new transition probabilities $T'(s,a,s')$ and rewards $Reward'(s,a,s')$ in terms of the old MDP such that the optimal values $V_{opt}(s) \forall s \in States$ are equal under the original MDP and the new MDP.

We can arrive at the result by first seeing what happens when we factor out gamma in the summation of

$$V_{opt}(s) = \max_{a \in \text{Actions}(s)} Q(s,a) \tag{19}$$

$$Q(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_{opt}(s')\right] \tag{20}$$

$$= \sum_{s'} \gamma T(s,a,s') \left[\frac{1}{\gamma} R(s,a,s') + V_{opt}(s')\right] \tag{21}$$

$$= \sum_{s'} T'(s,a,s') \left[R'(s,a,s') + V_{opt}(s')\right] \tag{22}$$

and voilá, γ is gone, if we define

$$T'(s,a,s') = \gamma T(s,a,s') \tag{23}$$

$$R'(s,a,s') = \frac{1}{\gamma} T(s,a,s') \tag{24}$$

where $s' \neq o$. In order to satisfy the constraint that $\sum_{s'} T'(s,a,s') = 1$, we define

$$T'(s,a,o) := 1 - \gamma \tag{25}$$

which gives us

$$\sum_{s'} T'(s,a,s') = \left(\sum_{s' \neq o} \gamma T(s,a,s') \right) + T'(s,a,o) \tag{26}$$

$$= (\gamma) + (1 - \gamma) \tag{27}$$

$$= 1 \tag{28}$$

Lastly, to ensure that the new state o does not impact V_{opt}, we define

$$R(s,a,o) := 0 \tag{29}$$

$$IsEnd(o) := \text{True} \tag{30}$$

which also implies that $V_{opt}(o) = 0$.

Problem 4: Learning to Play Blackjack

(b) *Now let's apply Q-learning to an MDP and see how well it performs in comparison with value iteration. First, call simulate using your Q-learning code and the identityFeatureExtractor() on the MDP smallMDP (defined for you in submission.py), with 30000 trials and default explorationProb.*

How does the Q-learning policy compare with a policy learned by value iteration (i.e., for how many states do they produce a different action)? (Don't forget to set the explorationProb of your Q-learning algorithm to 0 after learning the policy.) Now run simulate() on largeMDP, again with 30,000 trials. How does the policy learned in this case compare to the policy learned by value iteration? What went wrong?