CS 221 – Autumn 2018
Anand Natu
anatu - 06264867

Homework #4: Blackjack

Acknowledgements and Disclaimers

I collaborated with Nick Landy and Megan Knight in formulating my answers to this homework.

Problem 1

Generally, the transition probabilities are given by

State s	Action a	End state s'	Transition Probability T(s,a,s')
s	-1	s-1	80%
s	-1	s+1	20%
s	+1	s+1	30%
s	+1	s-1	70%

Problem 1a

The general value iteration formula for state s at iteration t is given by

$$V_{opt}^{(t)}(s) \leftarrow \max_{a \in Actions(s)} \sum_{s^t} Q_{opt}^{(t)}(s,a) = \max_{a \in Actions(s)} \sum_{s^t} T(s,a,s') \left[Reward(s,a,s') + \gamma V_{opt}^{(t-1)}(s')\right]$$

At the iteration t=0, we initialize the value function to be 0, and none of the information about the reward of the end states has propagated to the start state. So there is zero expected value to taking any action, meaning that the optimal value is simply

$$\boldsymbol{V_{opt}^{(0)}(s) = \left\{s: V_{opt}^{(0)}(s)\right\} = \{-2:0, -1:0, 0:0, 1:0, 2:0\}}$$

For any given state, we can either go forward or backward, but the expected reward $R_{E,a}$ for taking any given action a depends on the state we end in:

$$\max_{a \in Actions(s)} \sum_{s^t} T(s,a,s')[Reward(s,a,s')] = max\left(R_{E,+1}(s'), R_{E-1}(s')\right)$$

For the first iteration, we can cancel out the recursive term since all values are initialized to zero, so that we have

$$V_{opt}^{(1)}(s) = \max_{a \in Actions(s)} \sum_{s^t} T(s,a,s')[Reward(s,a,s')]$$

For s=-1 we have

$$V_{opt}^{(1)}(s=-1) = max([0.3(-5) + 0.7(20)], [0.8(20) + 0.2(-5)] = \max(12.5,15) = \mathbf{15}$$

For s=0 we have

$$V_{opt}^{(1)}(s=0) = max([0.3(-5) + 0.7(-5)], [0.8(-5) + 0.2(-5)] = \max(-5,-5) = \mathbf{-5}$$

And for s=1 we have

$$V_{opt}^{(1)}(s=0) = max([0.3(100) + 0.7(-5)], [0.8(-5) + 0.2(100)] = \max(26.5,16) = \mathbf{26.5}$$

So that we have

$$V_{opt}^{(1)}(s) = \left\{\boldsymbol{s}: \boldsymbol{V_{opt}^{(1)}(s)}\right\} = \{\mathbf{-2}: \mathbf{0}, \mathbf{-1}: \mathbf{15}, \mathbf{0}: \mathbf{-5}, \mathbf{1}: \mathbf{26.5}, \mathbf{2}: \mathbf{0}\}$$

For the second iteration, we repeat the computations and now must include the expected values of the previous iteration with the appropriate discount. However since $\gamma = 1$ we have

$$V_{opt}^{(2)}(s) = \max_{a \in Actions(s)} \sum_{s^t} T(s,a,s') \left[Reward(s,a,s') + V_{opt}^{(1)}(s')\right]$$

For s=-1 we have

$$V_{opt}^{(2)}(s=-1) = max([0.3(-5+(-5)) + 0.7(20+0)], [0.8(20+0) + 0.2(-5+(-5))] \\ = \max(11,14) = \mathbf{14}$$

For s=0 we have

$$V_{opt}^{(2)}(s=0) = max([0.3(-5+26.5) + 0.7(-5+15)], [0.8(-5+15) + 0.2(-5+26.5)] \\ = \max(13.45,12.3) = \mathbf{13.45}$$

For s=1 we have

$$V_{opt}^{(2)}(s=1) = max([0.3(100+0) + 0.7(-5-5)], [0.8(-5-5) + 0.2(100+0)] = \max(23,12) \\ = \mathbf{23}$$

So we have (states in **black**, optimum values in **blue**):

$$\boldsymbol{V_{opt}^{(0)}(s)} = \left\{\boldsymbol{s}: \boldsymbol{V_{opt}^{(0)}(s)}\right\} = \{\mathbf{-2}: \mathbf{0}, \mathbf{-1}: \mathbf{0}, \mathbf{0}: \mathbf{0}, \mathbf{1}: \mathbf{0}, \mathbf{2}: \mathbf{0}\}$$

$$V_{opt}^{(1)}(s) = \left\{\boldsymbol{s}: \boldsymbol{V_{opt}^{(1)}(s)}\right\} = \{\mathbf{-2}: \mathbf{0}, \mathbf{-1}: \mathbf{15}, \mathbf{0}: \mathbf{-5}, \mathbf{1}: \mathbf{26.5}, \mathbf{2}: \mathbf{0}\}$$

$$V_{opt}^{(2)}(s) = \left\{\boldsymbol{s}: \boldsymbol{V_{opt}^{(2)}(s)}\right\} = \{\mathbf{-2}: \mathbf{0}, \mathbf{-1}: \mathbf{14}, \mathbf{0}: \mathbf{13.45}, \mathbf{1}: \mathbf{23}, \mathbf{2}: \mathbf{0}\}$$

The above passes a test of intuition as well, as we have the reward from both end states propagating inward to raise the expected value of state 0, while the states closer to 2 decrease in iteration 2 compared to iteration 1, reflecting the information about the lower reward at end state -2 propagating right.

Problem 1b

The resulting optimal policy is simply the action leading to the maximum expected reward which is used to determine the optimal value, so that we have

Recall that for this problem, the arguments passed to our optimal value expressions for each state are represented as the expected reward of the forward and backward action:

CS 221 – Autumn 2018
Anand Natu
anatu - 06264867

$$V_{opt}^{(2)} = \max\left(R_{E,+1}, R_{E,-1}\right)$$

So that

$$\pi_{opt}^{(2)} = \arg\max\left(R_{E,+1}, R_{E,-1}\right)$$

For convenience, we will refer here to the -1 and +1 actions as backward and forward, respectively. Now we can figure out the maximizing arguments for the optimum value, which in turn constitute the optimum policy.

For s=-1 we have

$$\pi_{opt}^{(2)}(s=-1) = \arg\max(11,14) = -\mathbf{1}\ (\boldsymbol{backward})$$

For s=0 we have

$$\pi_{opt}^{(2)}(s=0) = \arg\max(13.45,12.3) = +\mathbf{1}\ (\boldsymbol{forward})$$

For s=1 we have

$$V_{opt}^{(2)}(s=1) = \arg\max(23,12) = +\mathbf{1}\ (\boldsymbol{forward})$$

Combining the above results in the optimum policy of

$$\boldsymbol{\pi_{opt}^{(2)}} = \{-\mathbf{1}: \boldsymbol{backward}, \mathbf{0}: \boldsymbol{forward}, \mathbf{1}: \boldsymbol{forward}\}$$

Problem 2

Problem 2b

If we have an acyclic graph, it means that what we have is essentially a standard dynamic programming problem, the only difference being that "costs" (which in the context of value iteration are interpreted as rewards, but computationally are treated the same) are weighted with the transition probabilities of traveling between any two nodes. To motivate this using the vocabulary of value iteration, an acyclic MDP implies that there is only one direction in the information from the end states propagates to the start, which again is identical to how a "vanilla" DP program is solved (beginning from the end states and traversing backwards acyclically). **Therefore, it follows that an acyclic MDP can be solved using standard dynamic programming and iterating through the states once.**

Problem 2c

Since the resultant value of the modified problem must be the same as the original problem, we require that the optimal value expression hold for both problems:

$$\begin{aligned} V'_{opt}(s) = V_{opt}(s) &= \max_{a \in Actions(s)} \sum_{s^t} T(s,a,s')\left[Reward(s,a,s') + \gamma V_{opt}(s')\right] \\ &= \max_{a \in Actions(s)} \sum_{s^t} T'(s,a,s')\left[Reward'(s,a,s') + \gamma' {V'}_{opt}(s')\right] \end{aligned}$$

Noting that $\gamma' = 1$ and expanding out the terms, we arrive at the following equations:

[1] $V'_{opt}(s) = V_{opt}(s)$

[2] $T(s, a, s')\gamma V_{opt}(s') = T'(s, a, s')V'_{opt}(s')$

[3] $T(s, a, s')R(s, a, s') = T'(s, a, s')R'(s, a, s')$

From [1] and [2] above, it follows that

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

Which by substitution into [3] yields

$$\boldsymbol{R'(s, a, s') = \frac{1}{\gamma} R(s, a, s')}$$

Then for the new state {o} in our modified problem, we must define a transition and reward $T'(s, a, o), R'(s, a, o)$ for this new state. Since we have demonstrated equivalence of optimal value above, it follows that there cannot be any reward at the new state:

$$\boldsymbol{R'(s, a, o) = 0}$$

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

Problem 4

Problem 4b

When running on smallMDP, out of the 27 total states that are defined for this problem, 22 of them overlap between value iteration and Q-learning while 5 differ. Conversely, when running on largeMDP, the overlap rate drops, with only 17 states being equivalent and 10 differing, i.e. the number of mismatches doubles for the larger MDP.

The reason why we see this drop in performance is because in the largeMDP, we have a larger state-space, i.e. there are more states (combinations of cards) to consider. Since the feature vector we used in these MDP problems is a contrived "identity" feature, the existing approach is tantamount to rote learning. Therefore, **this drop in performance happens with largeMDP because our existing Q-learning algorithm does not generalize – we are simply storing values of Q for every unique state-action pair, which becomes computationally expensive as the state-space grows larger.**

Problem 4d

Since the FixedRLAlgorithm we use does not actually do any learning, it will follow the exact policy that the value iteration reached on the originalMDP, but applied to the newThresholdMDP. Since the newThresholdMDP duplicates the state-space of originalMDP, but imposes a higher threshold, it follows that **the expected reward of running the FixedRLALgorithm on newThresholdMDP will be higher than the reward reached on the originalMDP.** This is because all of the actions are fixed by the policy, but since the threshold is higher there are actions which previously "went bust" but will now yield a reward.

Conversely, **running Q-learning on the newThresholdMDP directly yields a higher expected reward than the setup outlined above**. This is because the Q-learning algorithm is learning directly on this MDP, which has a higher threshold than the originalMDP on which value iteration was run. This means **the Q-learning is able to account for the fact that the modified MDP has a higher threshold, and as such can learn a new policy $\pi^{opt}_{Q-learning,newThresholdMDP} \neq \pi^{opt}_{ValueIteration,originalMDP}$ which takes advantage of the higher threshold and as such attains a policy which is optimized for the higher threshold.**