Blackjack - Stanford CS 221 Fall 2017-2018

Rohit Apte

2017-10-21

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

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

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

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

$$V_{opt}^{(t)}(s) = \begin{cases} 0 & \text{if isEnd(s)} \\ \max\limits_{a \in Actions} \sum_{s\prime} T(s, a, s\prime)[Rewards(s, a, s\prime) + \gamma V_{opt}^{(t-1)}(s\prime)] \end{cases}$$

Therefore
$V_{opt}^{(1)}(-2) = 0$
$V_{opt}^{(1)}(-1) = 15$
$V_{opt}^{(1)}(0) = -5$
$V_{opt}^{(1)}(1) = 26.5$
$V_{opt}^{(1)}(2) = 0$

$V_{opt}^{(2)}(-2) = 0$
$V_{opt}^{(2)}(-1) = 14$

$V_{opt}^{(2)}(0) = 13.45$
$V_{opt}^{(2)}(1) = 23$
$V_{opt}^{(2)}(2) = 0$

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

The optimal policy $\pi_{opt} = \begin{cases} 0 & \text{if isEnd(s)} \\ \max\limits_{a \in Actions} Q_{opt}(s,a) \end{cases}$
Therefore the resulting policy for non terminal states is
$\pi_{opt}(-1) = -1$
$\pi_{opt}(0) = 1$
$\pi_{opt}(1) = 1$

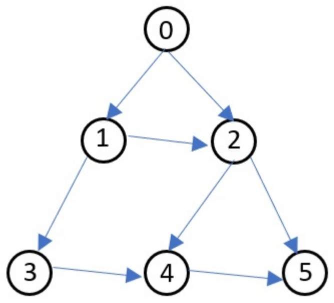
2 Problem 2: Transforming MDPs

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

a See submission.py

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 $(s, a, s\prime)$ triples.

Since the MDP is acyclic it can be represented as a Directed Acyclic Graph (DAG). Lets take a look at a particular example (see image)



Out states are $\{0,1,2,3,4,5\}$
We will define action a to go from state s to $s\prime$ to be $s\prime - s$.
$V_{opt}^{(t)}(5) = 0$ since its the end state
For states 4 and 3 there is only 1 action each so we dont need to do the max for V_{opt}
$V_{opt}^{(t)}(4) = T(4,1,5)[R(4,1,5) + \gamma V_{opt}^{(t-1)}(5)] = T(4,1,5)[R(4,1,5)]$
$V_{opt}^{(t)}(3) = T(3,1,4)[R[3,1,4) + \gamma V_{opt}^{(t-1)}(4)] = T(3,1,4)[R[3,1,4) + \gamma T(4,1,5)R(4,1,5)]$

For state 2 there are 2 possibilities so we need to do max on possible actions
$V_{opt}^{(t)}(2) = max(T(2,2,4)[R(2,2,4) + \gamma V_{opt}^{(t-1)}(4)], T(2,3,5)[R(2,3,5) + \gamma V_{opt}^{(t-1)}(5)])$
$= max(T(2,2,4)[R(2,2,4) + \gamma T(4,1,5)R(4,1,5)], T(2,3,5)R(2,3,5))$

As we can see the optimal policy V_{opt} is only a function of the transition probabilities and rewards of subsequent states (and not dependent on V_{opt} of past iterations. We therefore only need to calculate the transition probabilities $\times$ rewards for states in the correct order. If we take the topological sort of this DAG and reverse it, it gives us the correct ordering to calculate V_{opt} for each node with only a single pass over all the $(s, a, s\prime)$ triples.
The general algorithm for any acyclic MDP is as follows:

- For a node n in our MDP, define a graph starting at that node. So in our example for 2 we would have $2 \to 4 \to 5$ and $2 \to 5$ as our graph.
- Do a topological sort of the graph defined above. In our example at node 2, a valid sort would be $\{2,4,5\}$. Assume this returns a list l.
- Traverse the list in reverse order calculating the transition probabilities $\times$ rewards to that node. So for example for 5 we would have $T(2,3,5)R(2,3,5)$ and $T(4,1,5)R(4,1,5)$.
- Store the values in memory if required again.
- Keep going up the list until you get to the first element.

A valid topological sort for node 0 would be $\{0,1,3,2,4,5\}$.

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

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

Hint: If you're not sure how to approach this problem, go back to Percy's notes from the first MDP lecture and read closely the slides on convergence, toward the end of the deck.

We know that

$$V_{opt}^{(t)}(s) = \begin{cases} 0 & \text{if isEnd(s)} \\ \max\limits_{a \in Actions} \sum_{s\prime} T(s,a,s\prime)[Rewards(s,a,s\prime) + \gamma V_{opt}^{(t-1)}(s\prime)] \end{cases}$$

and

$V_{opt}^{(0)}(s) = 0 \forall s$

Therefore $V_{opt}^{(1)}(s) = \begin{cases} 0 & \text{if isEnd(s)} \\ \max\limits_{a \in Actions} \sum_{s\prime} T(s,a,s\prime)[Rewards(s,a,s\prime)] \end{cases}$

To understand the effect of $\gamma < 1$ on our data let us define a new end state o. Let us also assume that the probability reaching that state from any other state is $1 - \gamma$. Therefore the probability of not reaching that state when in any other state is γ. This is the same γ as the discount rate above. We need to adjust our transition probabilities by γ.

Thefore our transition probabilities $T\prime(s, a, s\prime) = \gamma T(s, a, s\prime)$.

To have the same V_{opt} we need to adjust the rewards to $\frac{Rewards(s,a,s\prime)}{\gamma}$. Now we have $V_{opt}^{(1)}(s) = \begin{cases} 0 & \text{if isEnd(s)} \\ \max\limits_{a \in Actions} \sum_{s\prime} T\prime(s,a,s\prime)[Rewards\prime(s,a,s\prime)] = \max\limits_{a \in Actions} \sum_{s\prime} \gamma T(s,a,s\prime)\frac{Rewards(s,a,s\prime)}{\gamma} \end{cases}$

Then

$$V_{opt}^{(t)}(s) = \begin{cases} 0 & \text{if isEnd(s)} \\ \max\limits_{a \in Actions} \sum_{s\prime} \gamma T(s,a,s\prime)\frac{Rewards(s,a,s\prime) + \gamma V_{opt}^{(t-1)}(s\prime)}{\gamma} = T\prime(s,a,s\prime)[Rewards\prime(s,a,s\prime) + V_{opt}^{(t-1)}(s\prime)] \end{cases}$$

Therefore if we have an MDP solver that can only handle discount factor of 1, when solving for $\gamma < 1$ we adjust the transiton probabilities and rewards as follows

$$T\prime(s,a,s\prime) = \begin{cases} (1-\gamma) & \text{if } s\prime = o \\ \gamma T(s,a,s\prime) & \text{otherwise} \end{cases}$$

$$R\prime(s,a,s\prime) = \begin{cases} 0 & \text{if } s\prime = o \\ \frac{1}{\gamma} R(s,a,s\prime) & \text{otherwise} \end{cases}$$

3 Problem 4: Learning to Play Blackjack

So far, we've seen how MDP algorithms can take an MDP which describes the full dynamics of the game and return an optimal policy. But suppose you go into a casino, and no one tells you the rewards or the transitions. We will see how reinforcement learning can allow you to play the game and learn its rules and strategy at the same time!

a See submission.py

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. 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 30000 trials. How does the policy learned in this case compare to the policy learned by value iteration? What went wrong?

On the small MDP Q-learning did a good job of learning the policy. There was only 1 state different out of 21 possible states. We have a threshold of 10 and the cards are 1,5 with multiplicity 2. It can learn when to take, peek or quit.

On the largeMDP Q-learning did much worse. There were 876 states different out of 2705 states. Here we have a high threshold (40) and the cards are 1, 3, 5, 8, 10 with multiplicity 3. It is not adequately able to learn when to take, peek or quit since there its easy to be over or under 40 with so many cards and small multiplicity.

c see submission.py

d Sometimes, we might reasonably wonder how an optimal policy learned for one MDP might perform if applied to another MDP with similar structure but slightly different characteristics. For example, imagine that you created an MDP to choose an optimal strategy for playing "traditional" blackjack, with a standard card deck and a threshold of 21. You're living it up in Vegas every weekend, but the casinos get wise to your approach and decide to make a change to the game to disrupt your strategy: going forward, the threshold for the blackjack tables is 17 instead of 21. If you continued playing the modified game with your original policy, how well would you do? (This is just a hypothetical example; we won't look specifically at the blackjack game in this problem.)

We know that
```
originalMDP = BlackjackMDP(cardValues=[1, 5], multiplicity=2, threshold=10,
peekCost=1)
newThresholdMDP = BlackjackMDP(cardValues=[1, 5], multiplicity=2, threshold=15,
peekCost=1)
```
Taking the optimal policy for originalMDP when applied to newThresholdMDP, it was still optimizing for threshold 10. With cardvalues=1 and 5 and 2 cards each it was typically quitting at 6, occationally hitting 10 or 7.
When we simulated Q-learning on newThresholdMDP instead, it started optimizing for threshold of 15. We often saw scores of 10, 11, 12 (12 being maximum allowed given the cards), some 9s (it must have peeked).
The rewards make sense since it is trying to optimize behavior to maximize reward and one of the conditions is it cannot breach the threshold or the reward is zero.