EECS 498: Reinforcement Learning
Homework 1 Responses

Tejas Jha
tjha

October 10, 2018

This document includes my responses to Homework 1 questions. Responses that involved the use of coding will provide references to specific lines of code to provide a better overview of how the problem was approached. The code can either be referenced in the Appendix or in the accompanied python script submitted with this assignment.

Question 1

Intuitively, it makes immediate sense why adding a constant to the reward function would not change the partial ordering since all rewards would be updated by the same constant value. Therefore, the optimal action to pick given a certain state would still be calculated to be the same as its value would still be larger than the value in choosing a sub-optimal action. However, this intuition is not a formal proof, so I will begin by first trying to define what the partial ordering in this situation is.

Value functions define a partial ordering over. From this, it can be said that for policies π and $\pi\prime$ and their corresponding value functions $v_\pi(s)$ and $v_{\pi\prime}(s)$, $\pi \geq \pi\prime$ if and only if $v_\pi(s) \geq v_{\pi\prime}(s)$ for all $s \in \mathcal{S}$. There is always at least one policy that is greater than or equal to all others, called the optimal policy and denoted as π_*. The optimal policies for a particular problem can be said to all share the same optimal state-value functions denoted by v_*, where $v_*(s) := \max_\pi v_\pi(s)$. By definition, v_* can be said to the be expected return for the best action from that state, which can be seen in equation (3.19) in the textbook as $v_*(s) = \max_a \sum_{s\prime,r} p(s\prime, r|s, a)[r + \gamma v_*(s\prime)]$

If we were to change the reward function to include an additional constant, then the above equation would result in a larger optimal value, but it would still require the use of the same action to maximize it. To see this, we can look at that for every action, the value of $p(s\prime, r|s, a)$ remains the same as it was before. The only change is inside the brackets where an additional c can be added. Every action state combination would result in a similarly distributed set of values, so any policy greater than or equal to another would also have a value greater than or equal to the other, thereby preserving the partial ordering.

Question 2

We have already shown in class that the policy iteration converges. Similarly, we can show that value iteration function converges and use that to estimate the number of steps needed. As a refresher, we were able to show this by defining some $\Delta_k := max_s|V_\pi - V_k|$. Using this definition, we know that $V_k(s) \leq$

$V_\pi(s) + \Delta_k$ and $V_k(s) \geq V_\pi(s) - \Delta_k$. We know from our value iteration that algorithm that:

$$V_{k+1}(s) = max_a \sum_{s\prime,r} p(s\prime, r|s, a)[r + \gamma V_k(s\prime)]$$

Using our previous inequalities, we can say that:

$$V_{k+1}(s) \leq max_a \sum_{s\prime,r} p(s\prime, r|s, a)[r + \gamma(V_\pi(s\prime) + \Delta_k)]$$

$$= max_a(\sum_{s\prime,r} p(s\prime, r|s, a)[r + \gamma(V_\pi(s\prime)] + \sum_{s\prime,r} p(s\prime, r|s, a)\gamma\Delta_k))$$

$$= V_\pi(s) + \gamma\Delta_k$$

Therefore, we can show that:

$$V_{k+1}(s) \leq V_\pi(s) + \gamma\Delta_k$$

And finally using our original definitions with the inequalities, we derive:

$$\Delta_{k+1} \leq \Delta_k$$

This proves that the algorithm will converge. In order to understand the number of iterations necessary, we need to look at our value for γ as this will affect the rate of convergence since each step follows the inequality above. The number of required steps will be somehow related to the value of γ. I looked over the paper found here: On the Complexity of Solving Markov Decision Problems. Following the paper, it would be possible to show that the number of iterations can be said the be bounded in the worst case by $1/(1-\gamma)log(1/(1/\gamma))$

Question 3

The value iteration update shows equation (4.10), which can seen to be of a form similar to the bellman optimality equations, but for a particular policy k and k + 1, where

$$v_{k+1} := \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s, A_t = a]$$

$$= \max_a \sum_{s\prime,r} p(s\prime, r|s, a)[r + \gamma v_k(s\prime)]$$

Similarly, we can write that:

$$q_{k+1} := \mathbb{E}[R_{t+1} + \gamma \max_{a\prime} q_k(S_{t+1}, a\prime)|S_t = s, A_t = a]$$

$$= \sum_{s\prime,r} p(s\prime, r|s, a)[r + \gamma \max_{a\prime} q_k(s\prime, a\prime)]$$

Question 4

Below are the plotted graphs as requested for this problem. As suggested, the top row represents the Epsilon-Greedy algorithm, the middle row represents the Optimistic Initial Value algorithm, and the bottom row represenst the UCB algorithm.

Epsilon-Greedy Starting with the cumulative regret plot, we can see that epsilon=0.01 starts off higher but its derivative decreases a lot faster, eventually touching underneath the epsilon=0.3 curve. This may result from more expoitation in the beginning time steps that resulted in more rewards at first, but eventually tapers down as it takes longer to explore other arms and approach the optimal solution. The epsilon=0.3 curve is much more constant in its slope. This would make sense since the agent in this situation would be performing more exploring that would increase the overall regret compared to lower epsilon values over time. The epsilon=0.1 curve performed the best as it had a lower regret due to more exploring earlier on than the epsilon=0.01 curve and more exploitation at later time steps once better expectations were calculated unlike the epsilon=0.3 curve.

The similar reasoning stated above can be used to describe why the plot for epsilon=0.1 reached a higher asymptotic average reward quicker than epsilon= 0.01. Also, it appears that epsilon=0.3 resulted in a curve with a lower asymptote for the averaged reward as it may continue to explore in later time steps when it can take advantage of calculated values and exploit.

The final graph for this algorithm shows the percent of optimal action taken over time step. It is a little difficult to see, but the epsilon=0.01 curve initally starts off higher, but tapers off earlier with a higher slope as it takes longer to explore. However, it appears that its slope may allow it to reach a similar or even higher asymptotic percent optimal action value after much more time elapses. The epsilon=0.1 curve once again appears to be reaching a higher asymptote than the epsilon=0.3 curve.

Overall, it appears that the perfectly ideal epsilon value would be as close to 0 as possible, but computational runtime limits how close we can approach 0 and still have a useful program.

Optimistic Initial Value In altering the initial values, epsilon was kept at 0, so the different runs all resulted in a greedy approach, but the optimistic initial value determined the rate of exploration initially. All the curves for all the graphs are very similar, but the curve for the inital value being set to 1 is slightly better for all three graphs (where it is slightly lower in the cumulative regret graph and slightly higher in the other two graphs). The results indicate that having optimistic values closer to the true values seems to perform best with little tradeoff as to increasing optimistic values. This makes some sense since an optimistic value closer to the true case would undergo less exploration and find the best arm to big greedily from. Comparing these graphs to the epsilon-greedy example, the graphs seem to definitely approach a higher average regward and percent optimal action much sooner due to the early exploration. A combination with a better choice of epsilon could allow for the graphs to reach a higher asymptote much sooner.

UCB The c=0.2 curve approached optmial actions much sooner than the other curves, but the c=1.0 curve appears to perform better in the long run as it eventually overtakes the c=0.2 curve in the averaged reward and percent optimal action graphs. The best feature of using UCB in comparison to the other algorithms appears to be in a slower increase int he cumulative regret. From the analysis of these graphs, it seems like it could be possible for the c=2.0 curve to approach the other curves, but it would definitely require a lot more time steps. In this case, the c values close to 1 appear to perform the best in te most reasonable amount of time.

From looking at these three different algorithms and their performances, not one parameter can be said to be the perfect solution. It appears to be an art of balancing the different parameters with possible combinations of them to emphisize the efficiancy of a desired attribute. For example, using UCB with smaller c values

prioritize lowering the cumulative regret, while epsilon values prioritize how quickly exploration goes on and how high the eventual averaged reward curve can go. Optimistic initalized values seem to help explore all possible actions at the very start to hopefully find an optimal action sooner. The Optimistic initialized values approach could then be said to be best when action rewards are fairly constant and do not vary widely over time. Epsilon-Greedy approaches seem to require higher epsilon values for more exploration should values vary a lot more.

Question 5

All relevant code can be found in the Appendix. The functions are called the same as they are in the homework descriptions.

******* Part (b) ********

Value Function for part (b):

[[3.17 8.93 4.28 5.31 1.4]

[1.4 2.88 2.11 1.81 0.48]

[0.03 0.7 0.63 0.34 -0.38]

[-0.88 -0.36 -0.29 -0.5 -1.06]

[-1.68 -1.19 -1.08 -1.25 -1.77]]

******* Part (c) ********

Value Function for part (c):

[[18.88 21.5 18.88 16.5 14.49]

[16.58 18.88 16.58 14.56 12.78]

[14.56 16.58 14.56 12.78 11.22]

[12.78 14.56 12.78 11.22 9.85]

[11.22 12.78 11.22 9.85 8.65]]

Map for part (c):

e nesw w nesw w

ne n nw w w

ne n nw nw nw

ne n nw nw nw

ne n nw nw nw

******* Part (d) ********

Value function and visualization of optimal policy for (d):

[[18.88 21.5 18.88 16.5 14.49]

[16.58 18.88 16.58 14.55 12.78]

[14.55 16.58 14.55 12.78 11.22]

[12.78 14.55 12.78 11.22 9.85]

[11.22 12.78 11.22 9.85 8.65]]

e nesw w nesw w

ne n nw w w

ne n nw nw nw

ne n nw nw nw

ne n nw nw nw

******* Part (e) ********

gamma = 1.0

[[10. 10. 10. 5. 10.]

[10. 10. 10. 10. 10.]

[10. 10. 10. 0. 10.]

[10. 10. 10. 10. 10.]

[10. 0. 10. 10. 10.]]

es nesw sw nesw s

nesw nesw nesw ew nesw

nesw nesw nesw nesw ns

nesw nesw nesw nesw nesw

n nesw ne nesw nw

Based on the optimal value function, and the printed policy, the optimal policy was definitely not acheived since there is no consensus as to which direction to go in with most states having equal probabilities for all actions.

gamma = 0.8

[[7.62 10. 7.62 5. 3.81]

[5.8 7.62 5.8 4.42 3.37]

[4.42 5.8 4.42 0. 2.57]

[3.37 4.42 3.37 2.57 1.96]

[2.57 0. 2.57 1.96 1.49]]

e nesw w nesw w

ne n nw w w

ne n nw nesw n

ne n nw w nw

n nesw n nw nw

Setting gamma to 0.8 brought us closer to the optimal policy, but there is still a path from state 4 that is said to go west to state 3, where B is. This would not be optimal since only a reward of 5 would result. Therefore, we can say that 0.8 is not yet providing an optimal policy.

gamma = 0.9

[[8.78 10. 8.78 5. 5.22]

[7.71 8.78 7.71 6.77 5.94]

[6.77 7.71 6.77 0. 5.22]

[5.94 6.77 5.94 5.22 4.58]

[5.22 0. 5.22 4.58 4.02]]

e nesw w nesw s

ne n nw w w

ne n nw nesw n

ne n nw w nw

n nesw n nw nw

With gamma = 0.9, we have reached our intention of getting an optimal policy. From all states, paths are assigned such that the optimal reward can be acheived. Notice that state 1, 3, 13, and 21 all have nesw because every action results in the game ending right after with the reward being a set value dependent on the starting location. However, for every other state, the paths are optimized so that they always lead to A at state 1, which would result in an eventual reward of +10.

******* Part (f) ********

We already know from the previous part that in this situation the critical value for gamma is between 0.8 and 1.0, and we have seen that the value of 0.9 does indeed provide an estimate that produces an optimal policy, and the value of 0.8 was close. The fact that 0.8 was close goes to show that the critical value at which the agent falters away from performing towards the intention is somwhere between 0.9 and 1.0, inclusive. We could either try to empirically find this critical value by testing different values at some precision, or mathematically derive the exact solution. Empirically finding it would involve a similar strategy as found in part e. If I had more time, I would explore solving it mathematically (will do after hw is due for fun :)).

Through performing the empirical experiments, I found that values even up to gamma= 0.995 found the optimal policy. Therefore, it appears that the critical value would be 1.0 for gamma (where I am assuming the critical value is to be the value at which the algorithm goes haywire and doesn't find the optimal policy).

Appendix: Relevant Code - tjha.py