RBE 595 — Reinforcement Learning
Midterm Exam

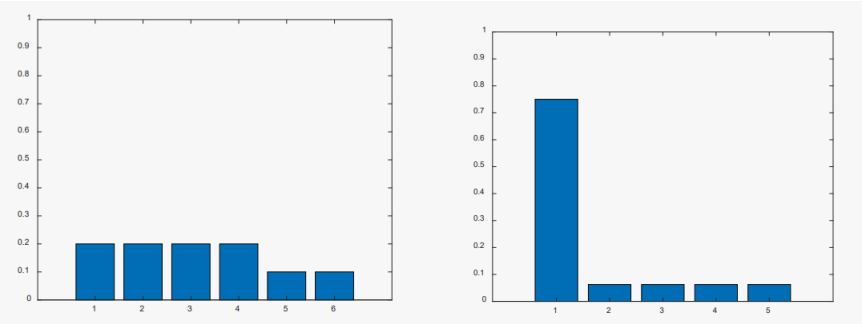
Arjan Gupta

Problem 1

Consider two random variables with distributions below:

$$p = \{0.2, 0.2, 0.2, 0.2, 0.1, 0.1\}$$

$$q = \{0.75, 0.0625, 0.0625, 0.0625, 0.0625\}$$



A. [4 points] Calculate the entropy for each variable.
B. [4 points] Intuitively how can you tell which variable has a higher entropy without calculating the entropy numerically? What does higher entropy mean?

Answer

A. The entropy for each variable is given by:

$$\begin{aligned} H(p) &= -\sum_{i=1}^{6} p_i \log_2 p_i \\ &= -\left(0.2\log_2 0.2 + 0.2\log_2 0.2 + 0.2\log_2 0.2 + 0.2\log_2 0.2 + 0.1\log_2 0.1 + 0.1\log_2 0.1\right) \\ &= -(4 * 0.2\log_2 0.2 + 2 * 0.1\log_2 0.1) \\ &= 2.5219 \end{aligned}$$

$$\begin{aligned} H(q) &= -\sum_{i=1}^{5} q_i \log_2 q_i \\ &= -\left(0.75\log_2 0.75 + 4 * 0.0625\log_2 0.0625\right) \\ &= 1.3112 \end{aligned}$$

B. Entropy is defined as the lack of expected information, or the 'surprise'/uncertainty of a random variable. We can tell that q has a higher entropy than p without calculating the entropy, because the histogram for q shows that there is a 'surprising' value of 0.75, which goes against the trend of the other values (which are all 0.0625). On the other hand, the histogram for p shows that all the values are fairly close to each other. In general, higher entropy means that the random variable has more uncertainty, so the likelihood of encountering a value closer to the expected value is lower.

Problem 2

2- [5 points] Which equation is correct? draw the corresponding backup-diagram and explain.

A. $v_*(s) = \sum_{r,s} \pi(a|s)p(s',r|s,a)[r + \gamma q_*(s')]$

B. $v_*(s) = \max_a q_*(s,a)$

C. $v_*(s) = \max_a \sum_{r,s} p(s',r|s,a)[r + \gamma q_*(s',a)]$

D. $v_*(s) = \max_a \sum_{r,s'} p(s',r|s,a)[r + \gamma v_*(s')]$

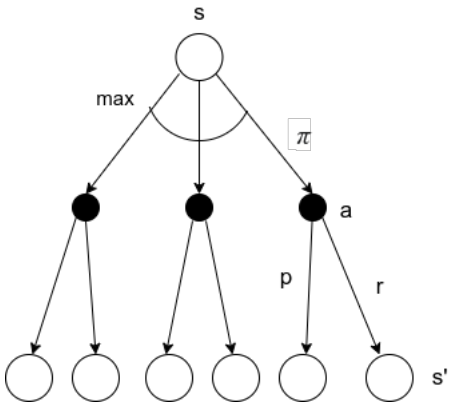
E. None of the above

F. B and D

Answer

Option F (B and D) is correct.

I have drawn the backup diagram for v_* as given below:



Explanation

v_* is the optimal state-value function. The backup diagram for v_* shows us that, if we start at a state s, we choose the action that maximizes the value of the state-action pair, and then we take the action. The action choice is taken using our current policy, π. Once we take the action, we end up in a new state, s', and we get a reward, r. The resultant state s' as well as the reward r are chosen according to the dynamics of the environment, $p(s', r|s, a)$, which is not something we can control. In summary, the optimal state value function chooses the action that maximizes the value of the state-action value.

Problem 3

Model-based RL methods suffer more bias than model-free methods. Is this statement correct? Why or why not?

Answer

This statement is correct. Model-based RL methods suffer more bias than model-free methods because the design of the model introduces bias. The model is a representation of the environment, and it is not possible to represent the environment perfectly. Therefore, the model will always introduce some bias.

Problem 4

Model-based RL methods are more sample efficient. Is this statement correct? Why or why not?

Answer

This statement is correct. Model-based RL methods can use a limited number of samples to learn a model of the environment. Given an episode from real-interaction, we can extract as much 'juice' as possible from it by using it to learn a model. Then, we can use the model to simulate more episodes, and use those episodes to improve the policy.

On the other hand model-free RL methods can only use the episode to improve the policy, so they require more 'samples' to learn the same amount.

Therefore, model-based RL methods are more sample efficient since they can learn more from the same number of samples.

Problem 5

What are the 4 steps of the MCTS algorithm? How does MCTS balance exploration and exploitation?

Answer

The 4 steps of the MCTS algorithm are:

1. **Selection**: Starting from the root node, we select a leaf node with an exploration-exploitation trade-off criteria (UCB1).
2. **Expansion**: We expand the leaf node by adding a child node for each possible action.
3. **Simulation**: We simulate an episode starting from the newly added child node. We use the roll-out policy to select actions during the simulation.
4. **Back-up**: We store action-values for each node in the tree.

Specifically, in the selection step, for balancing exploration and exploitation, we use the UCB1 formula:

$$UCB1(S_i) = \frac{v_i}{n_i} + C\sqrt{\frac{\ln N_i}{n_i}}$$

Where,
n_i is the number of times we have visited node S_i,
N_i is the number of times we have visited the parent of node S_i,
v_i is the value of node S_i, and
C is a constant that we choose.

The flow chart for the MCTS algorithm (tree policy) is shown below:
Now, let us describe how we use the above flow chart to balance exploration and exploitation. At first, we just have the root node, S_0. This technically currently a leaf node, but since it is the root-node, we make an exception and immediately add new child nodes to it, and rollout from the first new child node. Based on the results of the rollout, we update the value of the first new child node and its parent.

Then, we restart the algorithm from the root node. This time, since the root node is no longer a leaf node, we need to use UCB1 to select a leaf node. Now this is where the exploration-exploitation trade-off comes in. We will find that even though the first new child node has a high value (since we rolled out from it previously), it would be the choice in case of greedy selection. However, since we are using UCB1, we will instead consider the second new child node, which has a lower value, because its exploration term is higher. Specifically, the exploration term is higher because n_i is 0, making the second term in the UCB1 for that node infinitely large. This shows that in the planning phase, MCTS balances exploration and exploitation by using the UCB1 formula.

Problem 6

(Exercise 8.1, page 166) The nonplanning method looks particularly poor in Figure 8.3 because it is a one-step method; a method using multi-step bootstrapping would do better. Do you think one of the multi-step bootstrapping methods from Chapter 7 could do as well as the Dyna method? Explain why or why not.

Answer

Let us analyze how the n-step TD method would be applied to this problem.

Firstly, in n-step TD, the $G_{t:t+n}$ is given as,

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$$

One thing to notice is that, as given in the problem prompt, the reward for all actions is 0 except for the goal state, where the reward is 1. Therefore, we have two cases, depending on whether the goal state is reached within n steps or not.

If $t < T$, then $G_{t:t+n}$ is given as,

$$G_{t:t+n} = 0.95^n Q(S_{t+n}, A_{t+n})$$

If $t \geq T$, then $G_{t:t+n}$ is given as,

$$G_{t:t+n} = 0.95^{k-1} + 0.95^k Q(S_{t+n}, A_{t+n})$$

Where k is the number of steps taken before reaching the goal state (i.e. $k = T - t$).

And $Q(S_t, A_t)$ is given as,

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[G_{t:t+n} - Q(S_t, A_t) \right]$$

We can see that if we use 1-step TD, the algorithm's environment-interaction steps will be very similar to Dyna-Q with $n = 0$ planning steps.

However, I believe that in this particular problem, **the general n-step TD method will not perform as well as n-planning step Dyna-Q**, because it will take longer than Dyna-Q to learn the state-action values of the states leading up to the goal state. This is because, initially the state-action values of all the states will be 0. Then, the n-step TD method will only update the value of the state-action pair that encounters the goal state. All the states leading up to the goal state will simply have their value updated to 0. After a few episodes, the state-action values will be updated to non-zero values, but it will take longer than Dyna-Q.

On the other hand, Dyna-Q will use the simulated experience to update the value of all the state-action pairs that were encountered in the simulated experience. It will make good use of computational resources in its planning phase. With only 1 episode and a sufficiently large number of planning steps, Dyna-Q will be able to learn non-zero values for the state-action values leading up to the goal state.

Problem 6 continued on next page...

By the second or third episode, Dyna-Q will have learned the state-action values of the states leading up to the goal state much better than the n-step TD method. Perhaps after a high number of episodes, the n-step TD method will catch up to Dyna-Q, but it will be behind initially. Therefore, I think if we compare the general performance of the two algorithms for this particular problem, Dyna-Q will perform better than the n-step TD method.

Problem 7

(Exercise 8.2, page 168) Why did the Dyna agent with exploration bonus, Dyna-Q+, perform better in the first phase as well as in the second phase of the blocking and shortcut experiments?

Answer

In the first phase, as described in Example 8.2, the maze has an opening on the near side, and a wall on the far side. After 1000 time steps, the short path is blocked and the far side opening is created. The graph for the performance of Dyna-Q and Dyna-Q+ is shown below:
We can see that Dyna-Q+ performs better than Dyna-Q before and after the 1,000 time step mark. Before the 1,000 time step mark, Dyna-Q+ performs better because it is able to explore the grid-world more than Dyna-Q. This is because Dyna-Q+ uses the exploration bonus in its planning phase, given by $R = r + \kappa\sqrt{\tau(s,a)}$. This encourages the agent to find the far edge faster than Dyna-Q. After the 1,000 time step mark, Dyna-Q+ performs better again for the same reason. Since the short path is blocked, Dyna-Q+ is able to explore the grid-world faster than Dyna-Q.

In the second phase, as described in Example 8.3, the maze has an opening on far side, and after the 3,000 time step mark, the short path is additionally opened. The graph for the performance of Dyna-Q and Dyna-Q+ is shown below:
While the performance of Dyna-Q+ and Dyna-Q is similar before the 3,000 time step mark, we can see that around roughly the 4,000 time step mark, Dyna-Q+ begins to perform noticeably better than Dyna-Q. Here, the reward bonus is helping Dyna-Q+ create trajectories that lead to the short path. This is because the reward bonus is given by $R = r + \kappa\sqrt{\tau(s,a)}$, where $\tau(s,a)$ is the number of time steps since the last visit to state s after taking action a. Therefore, the reward bonus is higher for states that have not been visited in a long time. This encourages the agent to explore the grid-world more, and find the short path. The Dyna-Q agent does not have this advantage, and in fact never finds the short path, as noted by the book, "even with an ϵ-greedy policy, it is very unlikely that an agent will take so many exploratory actions as to discover the shortcut." So, exploration in the planning phase is what gives Dyna-Q+ the advantage over Dyna-Q.