Recitation 6
Reinforcement Learning

10-601: Introduction to Machine Learning

04/19/2019

1 Information Theory

Let's play a game: I roll two dices together, and you get paid $7 - (r_1 + r_2)$, i.e. how much lower the sum of two dice rolls is compared to 7, if the sum is higher than 7, you lose money. One of the dices is fair, however, the other is rigged so that it always rolls 4, 5 or 6 equally likely (and never below 4). Apparently, I'm not gonna tell you which dice is rigged.

1.How uncertain are you about your reward from this game? If your payoff is multiplied 1000 times (so if you roll a sum of 8, you lose 1000 instead of 1), does this change the uncertainty of your payoff?

2.Now if I roll the dices one by one, and you have seen the result from the first dice roll, does this reduce your uncertainty of your final payoff? By how much? Suppose there's an equal chance which dice I roll first.

3.What is the mutual information between these two events (knowing the sum of two dice rolls and knowing the first dice roll results)?

2 MDPs and the Bellman Equations

A Markov decision process is a tuple $(S, A, \{p(s'|s,a)\}, \gamma, R)$, where:

1. S is the set of states
2. A is the set of actions
3. $p(s'|s,a)$ is state the transition probabilities
4. $\gamma \in [0, 1)$ is the discount factor
5. $R : S \times A \mapsto R$ is the reward function

Dynamics of the MDP

1. We start in some state s_0

2. choose some action $a_0 \in A$
3. As a result of our choice, the state of the MDP randomly transitions to some successor state s_1. We drawn according to $s_1 \sim p(s'|s_0, a_0)$
4. Then we pick another action a_1, and so on.

We can represent it as

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

Sequence of states and payoff

Upon visiting a sequence of states $s_0, s_1, s_2, \dots$ with respective actions $a_0, a_1, a_2, \dots$ the **total payoff** is given by

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \dots$$

different sequences of states have a different payoff value.

Goal of reinforcement learning

Choose actions over time so as to **maximize** the expected value of the total payoff:

$$\mathbb{E}\left[R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \dots\right]$$

Things to notice:

- The reward at timestep t is discounted by a factor of γ^t. As $t \to \infty$ we have $\gamma^t \to 0$.
- To make this expectation large (max), we need to **accrue positive rewards as soon as possible** and postpone negative rewards as long as possible.
- We take the expectation of the total payoff, as to average its value over different sequences of states. Remember that different sequences of states have a different payoff value.

What about the policy function, value function and optimal value function?

- A **policy** is any function $\pi : S \mapsto A$ mapping from the states to the actions.

- We say that we are executing some policy π if, whenever we are in state s, we take action $a = \pi(s)$[1]
- For a given policy function π we can compute its **value function** as:

$$V^{\pi}(s) = \mathbb{E}\left[R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \ldots \mid s_0 = s, \pi\right]$$

  Notice that we take the expectation because, given that we start at state s, we have different sequence of observations all of them starting at the state s. In other words, we are interested in the expected value of the total payoff when we always start at state s according to the policy π.

- We also define the **optimal value function** according to

$$V^*(s) = \max_{\pi} V^{\pi}(s). \tag{1}$$

  In other words, this is the best possible expected total payoff that can be attained (when we always start at state s according to the policy π) using **any policy** π.

Why Bellman equations?

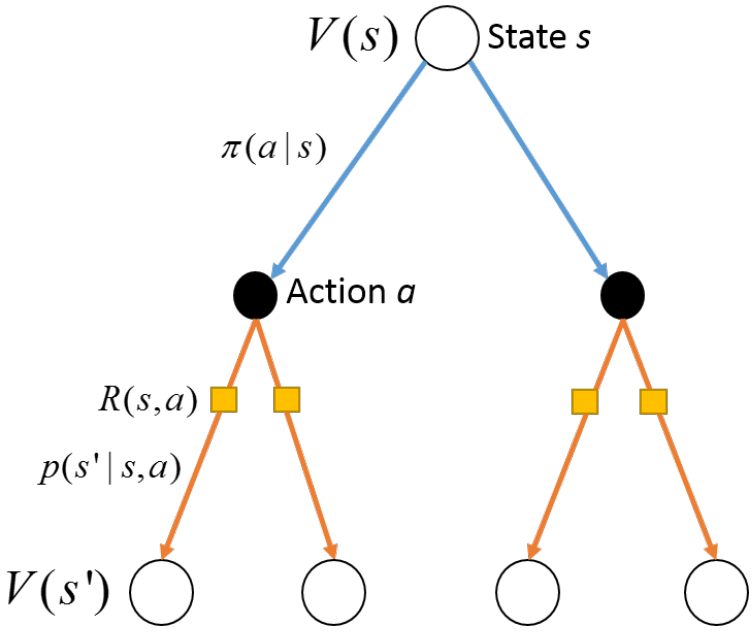


Figure 1: Visualization of Bellman Equation for $V(s)$

[1]This is for deterministic policy; for stochastic policy, $\pi(s)$ usually generates a probability distribution over all actions, from which we select an action based on some additional rules (random sampling, maximal sampling, ϵ greedy, etc. In stochastic cases, we generally write $a \sim \pi(s)$, or $Prob(a|s) = \pi(a|s)$.

Given a **fixed policy** π, its value function V^π satisfies the Bellman equations:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s' \in S} p(s'|s,a)(R(s,a) + \gamma V(s')).$$

This equation can be understood easily using backtracking in Fig.1:

- Going back from leaf $V(s')$ to black node a. Each $V(s')$ must be (i) discounted by γ (ii) added to immediate reward $r = R(s,a)$ and (iii) multiplied by transition probability, $p(s'|s,a)$;
- All leaf contribution, after previous step, aggregate at black node a, i.e., the sum over states s' in the equation above;
- To backtrack from node a to previous state s, each aggregated sum above must be multiplied by the policy $\pi(a|s)$.
- Finally, white node $V(s)$ aggregates results from previosu step, i.e., the first sum over action a in the equation above.

The interpretation of Bellman equation is also very intuitive: how valuable is the current state under a policy? Well, we take an action a under the policy, then get immediate reward $R(s,a)$ and the environment transitions to a new state s', of which the state value must be $V(s')$. Since the transition can be stochastic ($p(s'|s,a)$) and the value of the new state s' is only realized "in the future", we discount this "future value" and take its expectation under transition probabilities; since the action we take can also be stochastic (if policy is stochastic), we take another expectation of the sum of "immediate reward" and "expected future value" over all possible actions.

Question: The equations derived in this section (including Bellman optimality equations below) apply to both deterministic and stochastic policies, but equations under deterministic policy can be simplified a lot, how can you modify them for deterministic policy? Hint: how would Fig.1 change for deterministic policy? How can you simplify the equation even further when the transition is also deterministic?

Answer: Only one branch remains in the first level for deterministic policy, sum over a and $\pi(a|s)$ is gone, and $a = \pi(s)$ in the rest of the equation. For deteriministic transition, even the second sum is gone, and $s' = f(s,a)$ is a deterministic function for next state.

The **optimal value function** also has its own version of the Bellmans equations (named **Bellman optimality equation** for state value function), i.e.,

$$V^*(s) = \max_a \sum_{s' \in S} p(s'|s,a)(R(s,a) + \gamma V^*(s')).$$

compare this to Bellman equation for $V(s)$ before, the only change is the maximization operation in place for sum over action and policy $\sum_a \pi(a|s)$.

Question: Optimal value function should associate with an optimal policy, which should have its own $\pi(a|s)$ function, why is this function NOT present in the optimality equation for $V^*(s)$?

Answer: MDP can be proved to always have a deterministic optimal policy.

Question: What are the Bellman equation (and optimality equation) for action value function $Q(s, a)$? Use the visualization in Fig.2 to help your derivation.

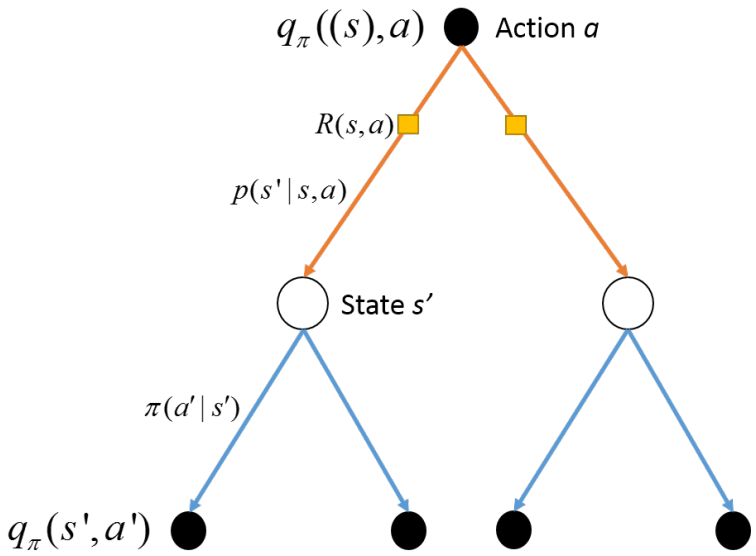


Figure 2: Visualization of Bellman Equation for Action Value Function, $Q(s, a)$

Answer:
Bellman equation:

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q^\pi(s',a').$$

Bellman optimality equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q^*(s',a').$$

Optimal policy

With the considerations above, we define the optimal policy $\pi^* : S \mapsto A$ as

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} p(s'|s,a)(R(s,a) + \gamma V^*(s')) = \arg\max_{a \in A} Q^*(s,a)$$

Note that π^* has the property that it is the **optimal policy** for all states s. Specifically, it is not the case that if we were starting in some state s then there would be some optimal policy for that state, and if we were starting in some other state s' then there would be some other policy that's optimal policy for s'. Specifically, the same policy π^* attains the

maximum in Equation (1) for all states s. This means that we can use the same policy π^* no matter what the initial state of our MDP is.

Question: How to solve the finite-state MDP?

Answer: Value iteration, policy iteration, even close form matrix solution

3 Value Iteration

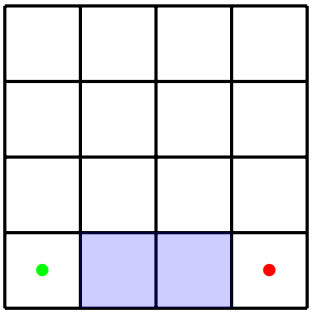
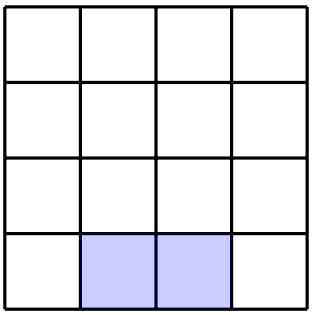


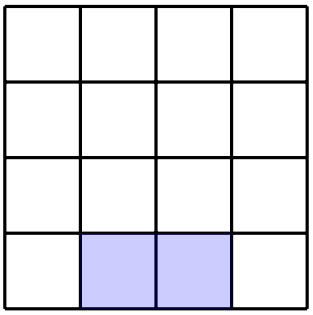
Figure 3: The cliff-walking environment

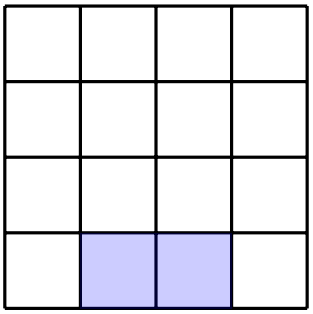
Here, we assume a 4x4 grid environment. The reward for reaching all cells is 0, except for the cell with the red dot, which has a reward of 1, and the shaded cells, which have a reward of -1[2]. Agent stops exploring if it meets either the shaded cell or red-dot cell. Assume an initial value (V) of 0 for each cell. The state space (S) is simply all the cells. The action space (A) is up, down, left or right. Our transition function is deterministic. If the agent tries to move out of the grid, it simply goes back to its previous state. The γ is 0.9. The initial state is the cell with the green dot.

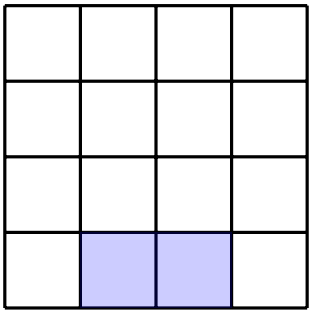
[2]One ambiguity in the "reward" here is whether we consider it as a function of (cell, action), or just a function of (cell) alone. The wording of the question takes the former interpretation. In recitation we'll talk about the latter as well, which may be more intuitive. $V(s)$ calculated using latter will be overall scaled by discount factor from $V(s)$ calculated using the former interpretation

1. Using value iteration, find the updated value of each cell for 4 iterations.

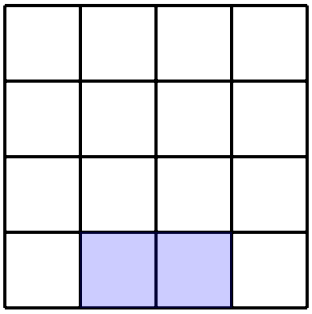








2. What is an optimal policy after you run value iteration to convergence?



3. Consider the process of policy improvement during value iteration. What is the entropy $H = 1/|S| \sum_s H(a|s)$ (average of action entropy at each cell) at the start of the value iteration (assuming we start with a random policy)? What is the entropy at the end of value iteration? In general, how does H change during value iteration?

4 Q-learning

Before we start discussing details about Mountain Car environment in the homework, let's recap the most important equation in Q-learning:

Question: What is the update equation for Q-learning? How is it different/related to Bellman equations for $Q(s, a)$? What advantage does Q-learning have over Policy/Value Iteration because of this difference?

In Mountain Car you control a car that starts at the bottom of a valley. Your goal is to reach the flag at the top right, as seen in Figure 4. However, your car is under-powered and can not climb up the hill by itself. Instead you must learn to leverage gravity and momentum to make your way to the flag. It would also be good to get to this flag as fast as possible.

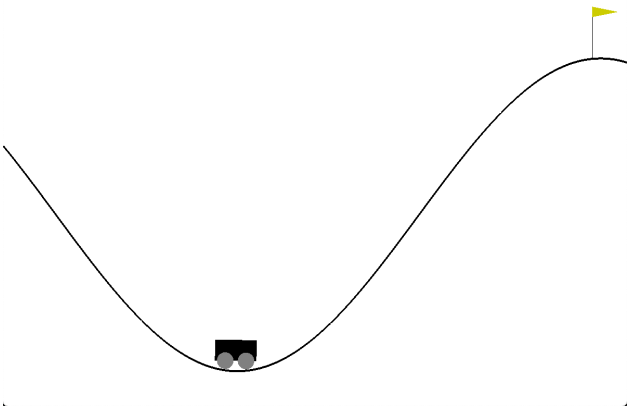


Figure 4: What the Mountain Car environment looks like. The car starts at some point in the valley. The goal is to get to the top right flag.

The state of the environment is represented by two variables, `position` and `velocity`. `position` can be between -1.2 and 0.6 inclusive and `velocity` can be between -0.07 and 0.07 inclusive. These are just measurements along the x-axis.

The actions that you may take at any state are $\{0, 1, 2\}$ which respectively correspond to (0) pushing the car left, (1) doing nothing, and (2) pushing the car right.

1. To solve Mountain car environment, would you use Value Iteration or Q-learning? Why?

2. Is Q-learning algorithm always linear approximator? By linear approximator we mean that $Q(s, a)$ is always a linear function of states and actions:

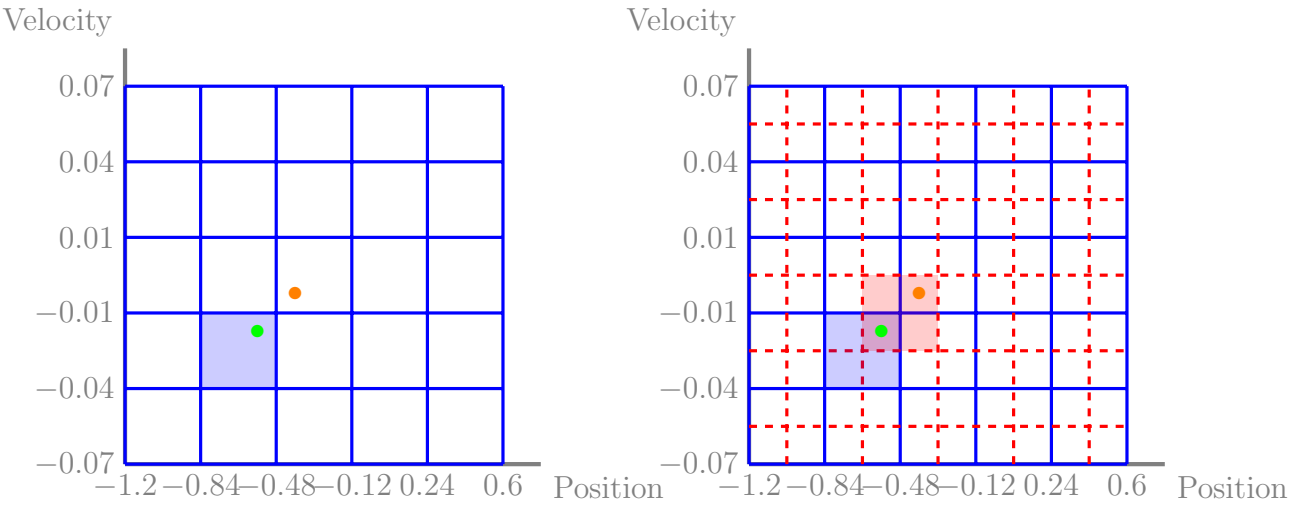
$$Q(s, a) = w^T \begin{bmatrix} s \\ a \end{bmatrix}$$

   If not, any ideas how could you learn a non-linear approximation?

3. If we solve the mountain car environment using Q-learning, we need to define a state representation. Is there any state representation you can think of?

4. **Discretization.** For the Mountain Car environment, we know that `position` and `velocity` are both bounded. What we can do is draw a grid over the possible `position-velocity` combinations as seen in Figure 5a. We then enumerate the grid from bottom left to top right, row by row. Then we map all states that fall into a grid square with the corresponding one-hot encoding of the grid number. For efficiency reasons we will just use the index that is non-zero. For example the green point would be mapped to $\{6\}$. This is called a *discretization* of the state space. **Why do we want to discretize the state space? Is there any downside to this approach?**

5. Instead of 2D discretization, we can draw two grids over the state space, each offset slightly from each other as in Figure 5b. **What are the two indices, one for each grid that map the green point?**



(a) A discretization of the state space of Mountain Car

(b) A tiling of the state space of Mountain Car

Figure 5: State representations for the states of Mountain Car

6. Think about the implementation of Q-learning algorithm on this environment, what are steps that need to be achieved? How many loops are needed? Can you improve the modularity of your code and separate the programming tasks into individual function/-classes?