CS 189/289A Introduction to Machine Learning
Fall 2023 Jennifer Listgarten, Jitendra Malik **DIS11**

1 Derivation of the Bellman Equations

This problem attempts to provide some intuitions for Markov Decision Process (MDP). The seemingly simple equations related to MDP sometimes hide subtleties, which we hope to illustrate through this exercise.

In an MDP, at each time step t, an agent receives a representation of the environment's state $S_t \in \mathcal{S}$, selects an action $A_t \in \mathcal{A}$, which leads the environment to produce a reward $R_{t+1} \in \mathcal{R}$ and a new state $S_{t+1} \in \mathcal{S}$. To set up an MDP, we need a few definitions. First, we define its dynamics, because we make the Markov assumption, it suffices to define the *one-step dynamics*:

$$p(s', r|s, a) = \mathbf{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]$$

Then, we define the *return* following time t as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$$

where $0 \leq \gamma \leq 1$ is the *discount rate*. We also define a *policy* that maps from states to the probabilities of actions:

$$\pi(a|s) = \mathbf{P}[A_t = a|S_t = s]$$

Finally, for all $s \in \mathcal{S}$, we define the *state-value function* of the state s under policy π as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

which represents the expected future return when starting at s and following π thereafter.

Similarly, for all $s \in \mathcal{S}, a \in \mathcal{A}$, the *action-value function* of taking the action a in state s under policy π is:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a]$$

A brief note on the notations: the notation $\mathbb{E}_\pi[X|Y = y]$ is used to indicate the expectation of random variable X conditioning on the random variable Y with value y, when following the policy π. Often, X relates to the reward. Implicitly, the expectation is taken over all variables that is random inside the expectation (except for the ones we condition on), which could include the state, the action and the reward (at the current and future time points), all of which could be influenced by the policy π. Derivations in MDP often involves manipulating this expectation.

(a) To familiarize ourselves with the notations, let's try to understand the relationships between these definitions by expressing the following:

- Express the expectation of R_{t+1} in terms of π and $p(s', r|s, a)$
- Express $v_\pi(s)$ in terms of q_π and π
- Express $q_\pi(s, a)$ in terms of v_π and $p(s', r|s, a)$

Solution: First, consider $\mathbb{E}_\pi[R_{t+1}|S_t = s]$. By definition of conditional expectation, we have:

$$\begin{aligned}\mathbb{E}_\pi[R_{t+1}|S_t = s] &= \sum_r rp(r|s) \\ &= \sum_r r \sum_{s',a} p(s', r|s, a)\pi(a|s)\end{aligned}$$

Note that even though we only denote one expectation, we are actually marginalizing out R_{t+1}, S_{t+1}, A_t.

Next, consider $v_\pi(s)$, we have:

$$v_\pi(s) = \mathbb{E}_\pi[q_\pi(s, a)] = \sum_a q_\pi(s, a)\pi(a|s)$$

Here we are using the law of total expectation. $q_\pi[s, a]$ is itself a conditional expectation. We introduce $A_t = a$ in the conditioning of the inside expectation (which marginalizes out S_{t+1}, R_{t+1} and all the future reward $R_{t+2}, R_{t+3}, ...$), then marginalize out A_t in the outside expectation. This is very similar to the previous subpart, except because of the recursive definition of G_t, we need to marginalize out all the future reward. But we don't need to do this explicitly since $q_\pi(s, a)$ already captures this.

Finally, consider $q_\pi(s, a)$, we have:

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a] = \sum_{s',r}[r + \gamma v_\pi(s')]p(s', r|s, a)$$

Here what changes is the order of expectation, where the outside expectation marginalizes out S_{t+1}, R_{t+1}, and the inside expectation marginalizes out $R_{t+2}, R_{t+3}, ...$ Note since we already conditioned on A_t, we don't marginalize it out.

(b) Next, we derive the Bellman equation, which demonstrates that a fundamental property of the value function is that it satisfies a certain recursive relationship. There are actually two related notions: the *Bellman expectation equation* and the *Bellman optimality equation*. Let's first derive the Bellman expectation equation (defined for all $s \in \mathcal{S}$):

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

Solution: We did all the work in the previous part. Combining subpart 2 and 3, we have

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

(c) In MDP, we are interested in finding the *optimal policy* which maximizes the value function. We define the *optimal state-value function* as the state-value function under the optimal policy π_*:

$$v_*(s) = \max_\pi v_\pi(s)$$

Similarity, the optimal action-value function as:

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

Let's derive the Bellman optimality equation (defined for all $s \in \mathcal{S}$):

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

Intuitively, this states that the value of a state under an optimal policy equals the expected return for the best action from that state.

Solution: First, note the optimal action-value function gives the expected return for taking action $A_t = a$ in state $S_t = s$ at time t and *thereafter* following an optimal policy. Using what we derived from Part (a), we have:

$$q_*(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] = \sum_{s',r}[r + \gamma v_*(s')]p(s', r|s, a)$$

It remains to be shown that

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

This is justified since $q_*(s, a)$ already accounts for the expected return for all time steps after t following the optimal policy, given the current state s and action taken a, so if we maximize $q_*(s, a)$ over the action $A_t = a$ taken at the current time, we obtain $v_*(s)$. In other words, if we have the optimal value function v_*, the actions that appear best after a one-step search will be the optimal actions.

2 Space Exploration

You are controlling a spacecraft on a mission to explore and gather data from various celestial bodies in a solar system. The spacecraft can be in one of three states based on its energy levels: 'FullEnergy', 'LowEnergy', and 'Depleted'. 'Depleted' is a terminal state, representing the spacecraft running out of energy and being unable to continue its mission. We denote the states as $\mathcal{S} = \{F, L, D\}$.

At each state (except "Depleted"), there are two possible actions: 'Conserve' and 'Explore'. 'Conserve' represents cautious exploration with energy conservation, while 'Explore' represents aggressive exploration consuming more energy. We denote the actions as $\mathcal{A} = \{C, E\}$.

The one-step dynamics of the system is defined as the following:

From 'FullEnergy':

1. Conserve (FullEnergy → FullEnergy): Probability = 1, Reward = 1 (safe exploration)
2. Explore (FullEnergy → LowEnergy): Probability = 0.5, Reward = 2 (risky but more rewarding exploration)
3. Explore (FullEnergy → FullEnergy): Probability = 0.5, Reward = 2 (successful aggressive exploration without losing much energy)

From 'LowEnergy':

1. Conserve (LowEnergy → FullEnergy): Probability = 0.5, Reward = 1 (successful energy conservation)
2. Conserve (LowEnergy → LowEnergy): Probability = 0.5, Reward = 1 (maintaining energy level)
3. Explore (LowEnergy → Depleted): Probability = 1, Reward = -10 (running out of energy)

In this problem, we will derive the policy iteration and value iteration updates for the MDP above. As a reminder, policy iteration consists of a *policy evaluation* step followed by a *policy improvement* step, defined as:

$$\text{Policy evaluation: } v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

$$\text{Policy improvement: } \pi'(s) = \operatorname*{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

Value iteration effectively combines policy improvement and a truncated policy evaluation:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

Note how the policy iteration and value iteration correspond to turning the Bellman expectation equation and the Bellman optimality equation into the respective updates rules.

(a) Suppose we initialize with a policy that always conserves regardless of the states, i.e. $\pi_0(C|s) = 1, \pi_0(E|s) = 0$ for all s. Also, we initialize value functions $v_0(s) = 0$ for all s. Let the discount rate $\gamma = 0.5$. Run policy iteration for two iterations. Does policy iteration converges after two iterations?

Solution: We start with policy evaluation:

$$\begin{aligned} v_1(F) &= p(F, r|F, C)[r + \gamma v_0(F)] \\ &= 1[1 + 0.5v_0(F)] = 1 \\ v_1(L) &= p(F, r|L, C)[r + \gamma v_0(F)] + p(L, r|L, C)[r + \gamma v_0(L)] \\ &= 0.5[1 + 0.5v_0(F)] + 0.5[1 + 0.5v_0(L)] = 1 \end{aligned}$$

where we abuse notation and uses r to denote the reward given the corresponding state and action.

Then, run policy improvement given the updated value functions:

$$\begin{aligned} \pi_1(F) &= \operatorname*{argmax}_{C,E} \{p(F, r|F, C)[r + \gamma v_1(F)], p(F, r|F, E)[r + \gamma v_1(F)] + p(L, r|F, E)[r + \gamma v_1(L)]\} \\ &= \operatorname*{argmax}_{C,E} \{C : 1[1 + 0.5v_1(F)], E : 0.5[2 + 0.5v_1(F)] + 0.5[2 + 0.5v_1(L)]\} \\ &= \operatorname*{argmax}_{C,E} \{C : 1.5, E : 2.5\} = E \\ \pi_1(L) &= \operatorname*{argmax}_{C,E} \{p(F, r|L, C)[r + \gamma v_1(F)] + p(L, r|L, C)[r + \gamma v_1(L)], p(D, r|L, E)[r + \gamma v_1(D)]\} \\ &= \operatorname*{argmax}_{C,E} \{C : 0.5[1 + 0.5v_1(F)] + 0.5[1 + 0.5v_1(L)], E : 1[-10 + 0.5v_1(D)]\} \\ &= \operatorname*{argmax}_{C,E} \{C : 1.5, E : -10\} = C \end{aligned}$$

We then run policy evaluation again given the updated policies. Note $\pi_1(F) = E \neq \pi_0(F), \pi_1(L) = C = \pi_0(L)$:

$$\begin{aligned} v_2(F) &= p(F, r|F, E)[r + \gamma v_1(F)] + p(L, r|F, E)[r + \gamma v_1(L)] \\ &= 0.5[2 + 0.5v_1(F)] + 0.5[2 + 0.5v_1(L)] = 2.5 \\ v_2(L) &= p(F, r|L, C)[r + \gamma v_1(F)] + p(L, r|L, C)[r + \gamma v_1(L)] \\ &= 0.5[1 + 0.5v_1(F)] + 0.5[1 + 0.5v_1(L)] = 1.5 \end{aligned}$$

Then run policy improvement given the updated values:

$$\begin{aligned} \pi_2(F) &= \operatorname*{argmax}_{C,E} \{C : 1[1 + 0.5v_2(F)], E : 0.5[2 + 0.5v_2(F)] + 0.5[2 + 0.5v_2(L)]\} \\ &= \operatorname*{argmax}_{C,E} \{C : 2.25, E : 3\} = E \\ \pi_1(L) &= \operatorname*{argmax}_{C,E} \{C : 0.5[1 + 0.5v_2(F)] + 0.5[1 + 0.5v_2(L)], E : 1[-10 + 0.5v_2(D)]\} \\ &= \operatorname*{argmax}_{C,E} \{C : 2, E : -10\} = C \end{aligned}$$

We have $\pi_2(s) = \pi_1(s)$ for all s, so the policy iteration has converged, and we have found the optimal policy: $\pi_*(F) = E, \pi_*(L) = C$.

(b) Run value iteration for two iterations. Does it converge after two iterations?

Solution: By definition of value iterations:

$$\begin{aligned}
v_1(F) &= \max\{p(F, r|F, C)[r + \gamma v_0(F)], p(F, r|F, E)[r + \gamma v_0(F)] + p(L, r|F, E)[r + \gamma v_0(L)]\} \\
&= \max\{1[1 + 0.5v_0(F)], 0.5[2 + 0.5v_0(F)] + 0.5[2 + 0.5v_0(L)]\} \\
&= \max\{1, 2\} = 2 \\
v_1(L) &= \max\{p(F, r|L, C)[r + \gamma v_0(F)] + p(L, r|L, C)[r + \gamma v_0(L)], p(D, r|L, E)[r + \gamma v_0(D)]\} \\
&= \max\{0.5[1 + 0.5v_0(F)] + 0.5[1 + 0.5v_0(L)], 1[-10 + 0.5v_0(D)]\} \\
&= \max\{1, -10\} = 1
\end{aligned}$$

Using these updated values, we can run another step of value iteration:

$$\begin{aligned}
v_2(F) &= \max\{p(F, r|F, C)[r + \gamma v_1(F)], p(F, r|F, E)[r + \gamma v_1(F)] + p(L, r|F, E)[r + \gamma v_1(L)]\} \\
&= \max\{1[1 + 0.5v_1(F)], 0.5[2 + 0.5v_1(F)] + 0.5[2 + 0.5v_1(L)]\} \\
&= \max\{2, 2.75\} = 2.75 \\
v_2(L) &= \max\{p(F, r|L, C)[r + \gamma v_1(F)] + p(L, r|L, C)[r + \gamma v_1(L)], p(D, r|L, E)[r + \gamma v_1(D)]\} \\
&= \max\{0.5[1 + 0.5v_1(F)] + 0.5[1 + 0.5v_1(L)], 1[-10 + 0.5v_1(D)]\} \\
&= \max\{1.75, -10\} = 1.75
\end{aligned}$$

After two rounds, value iteration hasn't converged yet.

(c) You might notice how these algorithms all look very similar to each other. This is because, as we have seen in the previous exercise, they are essentially just different angles of viewing the same relationship (or different ways of rewriting v and q in relation to each other and themselves, if you like). In practice, however, policy iteration and value iteration might have different convergence time, depending on the MDP and the specific implementations. What are some factors that might affect the convergence time of these algorithms?

Solution: Both policy iteration and value iteration are Dynamic Programming (DP) algorithms for solving MDP (i.e. finding the optimal policy and optimal value). Given the state space size $|\mathcal{S}|$ and action space size $|\mathcal{A}|$, the run times of these DP algorithms are polynomial in $|\mathcal{S}|$ and $|\mathcal{A}|$, which is much better than directing searching the policy space, since the total number of (deterministic) policy is $|\mathcal{A}|^{|\mathcal{S}|}$ (for each state, a deterministic policy must choose one of the $|\mathcal{A}|$ possible actions and the choice of action in each state is independent of the choice in other states).

In practice, these algorithms usually converge much faster than their theoretical worst-case run times. Initializations of the value functions and policies are important factors. Also, how the DP updates are actually implemented matters. Updates can be done out-of-place (updating the entire $v_{k+1}(s)$ vector using old values of $v_k(s')$, which is what we implemented above) or in-place (updating each entry of $v_{k+1}(s)$ using the newest value for $v(s')$), and they can be done synchronously (in each iteration, sweep over the entire state space) or asynchronously (update the values of states in any order whatsoever, using whatever values of other states happen to be available).