Module 5: Lesson 2

Dynamic Programming: The Q-function

Outline

- Q-function.
- Finding the optimal policies using the Q-function
- Application: Optimal selling time using the Q-function

The Q-function

The value function is a useful concept, but RL often considers an alternative tool to find the optimal actions.

Let $Q(s, a)$ denote the expected discounted future reward from starting in state s, taking a as our first action, and then continuing with some course of actions. Formally, the state-action value is given by,

$$Q(s, a) = \sum_{s'} \mathbb{P}(s'|s, a)\big[r(s', s, a) + \gamma \; Q(s', a')\big]$$

- The Q-value of being in state s and taking action a comes from the expected reward, $r(s', s, a)$, plus the expected discounted value of the future rewards, $\gamma \; Q(s', a')$.

In each state s', we need to know the value of behaving optimally, so that we reach the optimal function:

$$Q^*(s, a) = \sum_{s'} \mathbb{P}(s'|s, a)\big[r(s', s, a) + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a')\big]$$

Optimal actions

A convenient aspect of this approach is that knowing Q^*, it is easy to compute the optimal action in a state s:

$$\Pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a)$$

Compare this to the optimal policy that arises from using the value function:

$$\Pi^*(s) = \arg\max_{a \in \mathcal{A}} \sum_{s'} \mathbb{P}(s'|s, a)\big[r(s', a) + \gamma\ v^*(s')\big]$$

While both methods should reach the same optimal policy, by obtaining $Q^*(s, a)$, we get rid of the requirement of knowing or estimating all the potential transitions $\mathbb{P}(s'|s, a)$ across states.

While in DP setups we know perfectly the model, the advantage of this approach will become clear in later lessons with explicit RL applications, where $\mathbb{P}(s'|s, a)$ is "learned" from experience.

Iteration over the Q-function

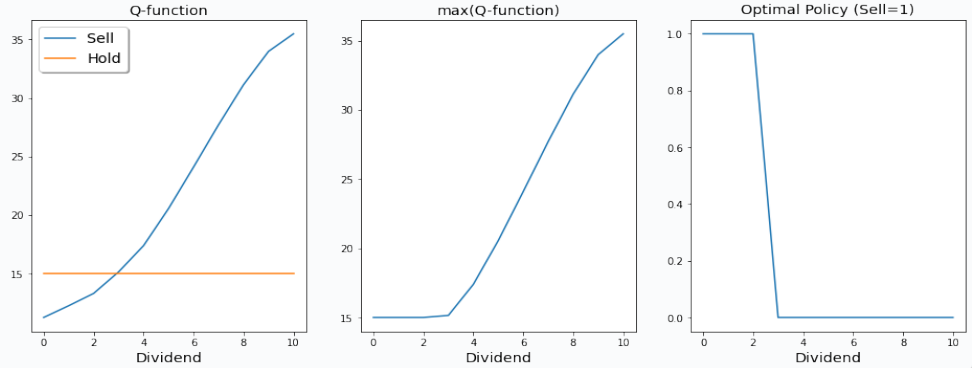
To compute Q^*, we can also follow an iteration algorithm. The pseudo-code for estimating the Q-function is as follows:

Begin with some flat guess, say $Q^{(0)}(s,a) = 0$, for all s, a.

1. Compute $Q^{(0,*)}(s) = \max_{a \in \mathcal{A}} Q^{(0)}(s,a)$.
2. Compute $Q^{(1)}(s,a) = \sum_{s'} \mathbb{P}(s'|s,a)\big[r(s',s,a) + \gamma\ Q^{(0,*)}(s')\big]$.
3. Stop if $Q^{(0)}$ and $Q^{(1)}$ are close enough. Otherwise, set $Q^{(0)} = Q^{(1)}$ and return to step 1.

Application 1: Optimal selling time

In the Jupyter Notebook for this lesson, we use the Q-function to determine the optimal "time" to sell an asset that we developed in Lesson 1. Notice that, despite the somewhat different approach, the outcomes are effectively the same as in Lesson 1.



Application 2: The gambler's problem

In the Jupyter Notebook for this lesson, we also present another application of DP in the context of a gambling problem.

The problem features an agent that must make a sequence of bets on the outcomes of a flipping coin until reaching a wealth target or running out of money.

In the Jupyter Notebook, we show how to solve for the optimal bets, i.e., those that maximize the probability of reaching the wealth target.

- Notice that the optimal actions do not have to be unique, although forcefully the optimal value must be unique.

You can play around with the parameters of the model to understand how DP methods reach different optimal choices.

Summary of Lesson 2

In Lesson 2, we have looked at:

- Definition of the Q-function
- Iteration over the Q-function

$\Rightarrow$ **References for this lesson:**

Sutton, R. S., Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 2018. (see Chapters 3 & 4)

TO DO NEXT: Now, please go to the associated Jupyter Notebook for this lesson to study further examples of applications of dynamic programming using the value function and Q-function.

In the next lesson, we will build on one further method to find the optimal policies, using policy iteration.