Module 5: Lesson 1

Reinforcement Learning: Introduction to Dynamic Programming

Outline

- Reinforcement Learning and Dynamic Programming
- The value function and value iteration
- Application: Optimal selling time

Reinforcement Learning and Dynamic Programming

In Reinforcement Learning (RL), an agent makes observations and takes actions within an environment, and in return, it receives rewards.

- The agent's objective is an optimal control problem, where the objective is to learn to act in a way that will maximize its expected long-term rewards.

In Module 5, we are going to build on the tools of Dynamic Programming (DP):

- DP refers to a collection of algorithms that can be used to compute optimal policies given perfect knowledge of the environment.

Classical DP algorithms are of limited utility in RL both because of their assumption of perfect knowledge and of their computational requirements.

- However, DP methods are an important foundation for RL algorithms, which attempt to achieve the same objective as a classical DP problem under relaxed assumptions.

Value: A measure of long-term rewards

We defined a RL setup as a process where, at each time step t, the environment is represented by a state $s_t \in \mathcal{S}$, and an agent taking an action $a_t \in \mathcal{A}$ that implies (i) a reward $r_t \in \mathcal{R}$ for the agent, and (ii) a transition to state $s_{t+1} \in \mathcal{S}$.

The problem boils down to finding a policy $\Pi : \mathcal{S} \to \mathcal{A}$ that maps states into actions.

What is the criterion for a good policy? We are going to consider the concept of **value**:

$$v_t(s) = \sum_{s'} \mathbb{P}(s'|s,a)\left[r_{t+1}(s',s,a) + \gamma \; v_{t+1}(s')\right] \tag{1}$$

where "$'$" denotes the next period's choices and states.

Given a choice now of action a, the value for an agent in state s comes from its reward out of action a and the value in future states, considering that the choice a may also affect the transition to s'.

- An important assumption in DP is that we know the transition across states and how actions affect those transitions, $\mathbb{P}(s'|s,a)$.

The parameter $\gamma \in (0,1)$ captures the extent to which current rewards are more relevant relative to more distant ones (patience, or *discount factor*).

Value: Stationary environment

Although we are going to cover a wide range of problems, we first analyze stationary problems with infinite-horizon:

- The agent will take actions forever, albeit possibly with some terminal state(s).
- In a stationary setup, whenever the agent has stepped into state s, the agent faces the same problem as it was on the previous time it reached s, and so the behavior is the same. Thus, we can drop the subscript t from the value function.

The best policy would lead to the function $v^*(s)$:

$$v^*(s) = \max_{a \in \mathcal{A}} \sum_{s'} \mathbb{P}(s'|s,a)\big[r(s',s,a) + \gamma\ v^*(s')\big] \tag{2}$$

In every state, we want to choose the action that maximizes the value of the future. Banach's fixed-point theorem guarantees that there exists a single optimal value $v^*(s)$ for each s.

The value iteration algorithm

One way to compute $v^*(s)$ is to solve a system of equations using linear algebra. However, the most practical way is to exploit the following **value iteration** algorithm:

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

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

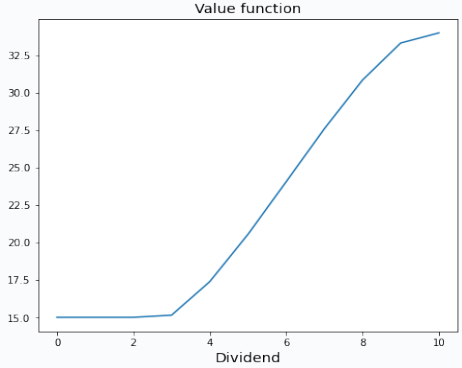
It is guaranteed that we can get close enough to v^* within a finite number of iterations.

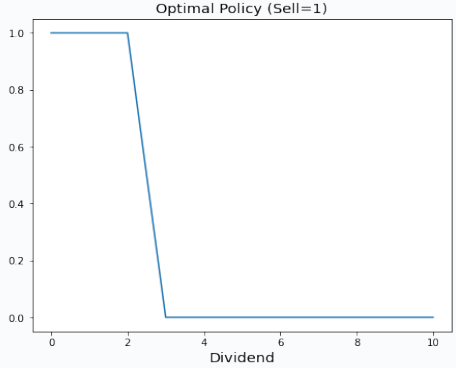
The optimal policy is then:

$$\Pi^*(s) = \arg\max_{a \in \mathcal{A}} \sum_{s'} \mathbb{P}(s'|s,a)\big[r(s',a) + \gamma\ v^*(s')\big] \tag{3}$$

Application: Optimal selling time

In the Jupyter notebook for this lesson, we implement the value function iteration to determine the optimal "time" to sell an asset.





Summary of Lesson 1

In Lesson 1, we have looked at:

- The basics of Dynamic Programming and the definition of value
- The value iteration algorithm

⇒ **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 implement the value function iteration algorithm.

In the next lesson, we will build an alternative method to find the optimal policies, using the Q-function.