Finite Markov Decision Problems

Diego Ascarza-Mendoza

Escuela de Gobierno y Transformación Pública

Outline

Introduction

- In this session, we introduce the formal problem we try to solve in the rest of the course.
- This problem involves evaluative feedback, but also choosing different actions in different situations.
- MDPs involve delayed reward and the need to tradeoff immediate and delayed reward.
- In bandit problems, we estimated $q_*(a)$ of each a. In MDPs we estimate the value of each action a in each state s $q_*(s, a)$.
- State-dependent quantities are essential to accurately assigning credit for long-term consequences to individual action selections.

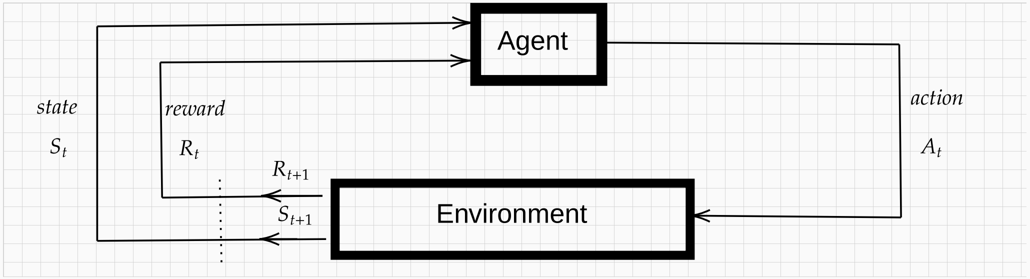
Introduction

- MDPs are idealizations of the RL problem where theoretical statements can be made.
- We will learn concepts such as returns, value functions, and Bellman equations.
- We will see that, as in any part of AI, there is tension between the breadth of applicability and mathematical tractability.

Outline

The Agent-Environment Interface

Figure 1: The agent-environment interaction in a MDP



The Agent-Environment Interface

- The learner and decision maker is called **the agent**.
- The thing it interacts with, everything outside the agent is called **the environment**.
- The agent interacts with its environment selecting **actions** and the environment responds to these actions and presents new situations to the agent.
- The environment also gives rise to rewards (numerical values), that the agent seeks to maximize over time by choosing actions.

The Agent-Environment Interface

- Formally speaking, agent and environment interact in a sequence of discrete time steps $t = 0, 1, 2, 3, ...$
- At each t, agent receives representation of envrionment's state, $S_t \in \mathcal{S}$.
- Based on state, agent selects action $A_t \in \mathcal{A}(s)$.
- One step later, as a consequence of tis action, agent receives numerical reward $R_{t+1} \in \mathcal{R} \in \mathbb{R}$, and finds itself in a new state S_{t+1}.
- MDP and agent together give rise to a sequence:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ... \tag{1}$$

- In *finite* MDP, $\mathcal{S}, \mathcal{A}$, and $\mathcal{R}$ are finite.

The Agent-Environment Interface

- In finite MDPs, R_t and S_t have well-defined discrete probability distributions that depend only on the preceding state and action.
- Formally, for particular values $s' \in \mathcal{S}$ and $r \in \mathcal{R}$, there exists a probability of those values occurring at t, given preceding state and action:

$$p(s', r|s, a) \doteq Pr\left\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\right\}, \tag{2}$$

  $\forall s', s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$.
- The function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ defines the *dynamics* of the MDP.
- p is an ordinary deterministic function that specifies a probability distribution for each choice of s and a:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \quad \forall \quad s \in \mathcal{S}, a \in \mathcal{A}(s). \tag{3}$$

The function p

- The probabilities given by p completely characterize the environment's dynamics.
- In other words, the probability of each possible value for S_t and R_t depends only on the preceding state and action S_{t-1} and A_{t-1} **(not at all on the whole history)**.
- This is a restriction on the state. The state must include information about all aspects of the past-agent interaction that matter for the future.
- When this condition holds, the state is said to have the *Markov Property*. We will work under this assumption throughout this course.

The function p

- From the function p, we can compute anything else one might want to know about the environment. For instance, the *state-transition probabilities*: $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0,1]$,

$$p(s'|s,a) \doteq Pr\left\{S_t = s'|S_{t-1} = s, A_{t-1} = a\right\} = \sum_{r \in \mathcal{R}} p(s',r|s,a). \tag{4}$$

- We can also compute the expected rewards for state-action pairs as a function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s,a) \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s',r|s,a), \tag{5}$$

- Also thee expected rewards for state-action-next state triples: $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$:

$$r(s,a,s') \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s',r|s,a)}{p(s'|s,a)}, \tag{6}$$

For most of our purposes, we will be using 2, but having the other notations is sometimes convenient.

Some remarks

- MDP framework is abstract and flexible: can be applied to many different problems in many ways.
- For instance, time steps can mean fixed intervals of real time, or to arbitrary successive stages of decision making and acting.
- Actions can be things like voltages applied to the motors of a robot arm, or deciding whether to go to grad school or not.
- States can also take different forms: they can be low-level sensations (sensor readings) or they can be symbolic descriptions of objects in a room. States can also be entirely mental or subjective.
- Even actions can be mental or subjective, such as what to think about or where to focus attention.

About the agent-state boundaries

- The boundary between agent and environment is typically not the same as the physical boundary of a robot's or animal's body.
- The boundary is usually drawn closer to the agent than that.
- For instance, motors and mechanical linkages of a robot should usually be considered parts of the environment rather than parts of the agent.
- In an MDP framework applied to a person, muscles, and skeleton inside the physical body are considered external to the agent.
- **General rule:** Anything that can't be changed arbitrarily by the agent is considered to be part of its environment.

About the agent-state boundaries

- We do not assume that everything in the environment is unknown to the agent.
- However, the reward computation is considered to be computed externally to the agent. It defines the task facing the agent. It has to be beyond its ability to change arbitrarily.
- Even if you fully know your environment, the RL can be challenging: Rubik's Cube.
- MDP framework is a considerable abstraction of the problem of goal-directed learning from interaction. It proposes that any problem of learning goal-directed behavior can be reduced to three signals passing back and forth between the agent and its environment:
    1. Actions
    2. States (bases on which actions are made).
    3. Rewards (defines the agent's goal).

Example

- Consider reinforcement learning is being applied to an individual who needs to determine how to allocate his wealth in terms of how much to save and how much to consume.
- Let's say that he only has bonds as the only saving asset available.
- In each period, wealth can be used to buy bonds or to consume.
- The action in this setting might be saving rates that allow for assigning a certain level of consumption and allow also to save resources for tomorrow.
- Let's say that the individual is subject to productivity shocks. Sometimes he is productive, sometimes he is not.
- Our states here are: the stock of capital and the state of productivity. Usually states and actions are lists or vectors, while returns are always a single number.

Outline

Goals and Rewards

- In RL, the goal of the agent is formalized in terms of a signal called *reward*.
- At each time step, the reward is a simple number, $R_t \in \mathbb{R}$.
- Agent wants to maximize the total reward it receives: cumulative reward in the long run!
- Think of this as, you want to maximize your lifetime happiness, you don’t only care about what will happen next weekend.
- Reward signals have been proven to be flexible and widely applicable. You don’t believe me?

Examples

- Make a robot learn to find and collect empty soda cans for recycling.
- One can give a reward of zero most of the time, and then a reward of $+1$ for each can collected.
- One might also want to give the robot negative rewards when it bumps into things.
- Or for instance, if you want to learn how to play chess, the natural rewards are $+1$ for winning, -1 for losing, and 0 for drawing.
- **The agent always learn to maximize its reward**: If we want it to do something for us, we must provide rewards in a way that the agent maximizes them so it achieves our goals.

To keep in mind about rewards

- The reward signal is not the place to impart to the agent prior knowledge about how to achieve what we want it to do.
- For instance, in chess, all you want is to win, and you only get a reward if you win. The subgoals should not be a reward, as it deviates from the main goal.
- The reward signal is a signal of communicating to the robot what you want it to achieve, not *how* you want it achieved.

Outline

Returns and Episodes

- Let's formalize our learning objective now. We want to maximize cumulative reward in the long run.
- If the sequence of rewards after time step t is denoted $R_{t+1}, R_{t+2}, \ldots$, we seek to maximize the *expected return*, which we denote by G_t.
- G_t is a specific function of the reward sequence. The simplest case is:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T, \tag{7}$$

- T is a final time step. This makes sense when there is a natural notion of final time step: agent-environment interaction breaks naturally into subsequences (*episodes*).
- This includes plays of a game or any repeated interaction. Each episode ends in a state called the *terminal state*.
- After the terminal state, there is a reset to a standard starting state or a sample from a standard distribution of starting states. The new episode begins independently of how the previous one ended.

Returns and Episodes

- Tasks with episodes of this kind are called *episodic tasks*. We denote the nonterminal states by $\mathcal{S}$ and the set of all states plus the terminal state by $\mathcal{S}^+$.
- The time of termination, T, is a random variable that varies from episode to episode.
- In many cases, the agent-environment interaction does not break naturally into episodes (continues without limit).
- This happens when you have an ongoing process-control task, or for instance, if you have a robot with a long life span. These are called *continuing tasks*.
- The return in 7 is problematic because $T = \infty$: what we maximize could be infinite.
- To avoid this issue, we use something called *discounting*.

Returns and Episodes

- With discounting, agent selects action to maximize *expected discounted return*:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{8}$$

- $\gamma \in [0, 1]$ is the *discount rate*.
- The discount rate determines the present value of future rewards: a reward received in k steps in the future is worth only γ^{k-1} what it would be worth if received immediately.
- If $\gamma < 1$ and $\{R_k\}$ is bounded, then 8 is finite.
- If $\gamma = 0$ the agent is myopic.

Returns and Episodes

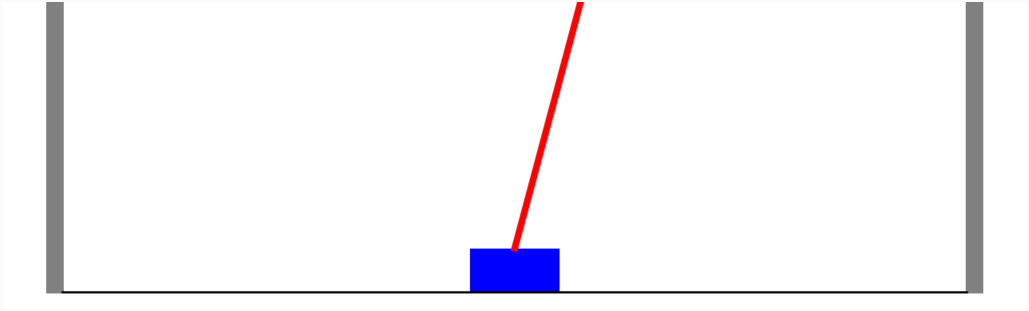
- When the agent is myopic, he only cares about choosing A_t to maximize R_{t+1}.
- If each agent's actions only influence the immediate reward, a myopic agent could maximize 8 by separately maximizing each immediate reward.
- In general settings, maximizing immediate reward can reduce access to future rewards (consumption/saving).
- Returns at successive time steps are related in a useful way for algorithms:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = R_{t+1} + \gamma G_{t+1}, \tag{9}$$

- This works $\forall t < T$. Even if termination occurs at $t+1$, we define $G_T = 0$.

Pole-Balancing example

Figure 2: Pole-Balancing



Pole-Balancing example

- We could frame this problem in many ways. Reward could be $+1$ for every time step on which failure did not occur.
- We could use discounting to avoid having ininifte return.
- We could also have a reward of -1 on each failure and zero otherwise. In this case, the return would be $-\gamma^K$ where K is the number of steps before failure. Success: keeping the pole balanced as long as possible.

Unified Notation for Episodic and Continuing Tasks

- We want a unified notation for episodic tasks and continuing tasks.
- With episodic tasks, we need to consider a series of episodes, each of which consists of a finite sequence of time steps.
- So, the state representation in time-step t in episode i strictly speaking is given by $S_{t,i}$. The same applies to $R_{t,i}, \pi_{t,i}, T_i$, etc.
- Most of the time, we don't have to distinguish between episodes, so we will omit the subscript i.
- We also need a convention to obtain a single notation that covers both episodic and continuing tasks.
- We can merge them by considering episode termination to be the entering of a special *absorbing state* that generates only rewards of zero.

Outline

Policies and Value Functions

- RL algorithms imply most of the time estimating *value functions*.
- Value functions are functions of states or state-action pairs that estimate *how good* it is for the agent to be in a given state or to perform an action in a certain state.
- How good means how much future returns can be expected. Rewards of course depend on actions taken by the agent.
- Value functions are defined **with respect to** a particular way of acting: policies.
- A *policy* is a mapping from states to probabilities of selecting each possible action.
- If agent follows a policy π in time t, $\pi(a|s)$ is probability that $A_t = a$ if $S_t = s$.
- RL specifies how the agent's policy is changed as a result of its experience.

Value Functions

- The *value function* of a state s under a policy π, denoted by $v_\pi(s)$ is given by:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s\right], \quad \forall s \in S, \tag{10}$$

- $\mathbb{E}_\pi$ is the expectation given that the agent takes policy π. The value function tells us the expected return when starting in s and following π thereafter.
- v_π is the *state-value function for policy* π.
- We can also define the value of taking action a in state s under policy π, denoted by $q_\pi(s,a)$:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{11}$$

- This tells you the value of taking action a in state s under a policy π

VALUE FUNCTIONS

- The value functions v_π and q_π can be estimated from experience.
- If an agent follows policy π and maintains an average, for each state encountered, of the actual returns that followed that state, this will converge to the state's value, $v_\pi(s)$, as the number of times that state is encountered $\rightarrow \infty$.
- If separate averages are kept for each action taken in each states, these will also converge to $q_\pi(s, a)$.
- Estimation methods of this kind are called *Monte Carlo methods*. If there are many states, of course, these methods are not practical.
- Instead, the agent could have v_π and q_π as parameterized functions and ajust parameters to better match the observed returns (approximations).

Recursive Representation

- A fundamental property of value functions is that they satisfy recursive relationships. For any policy π and any state s, the following condition holds:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s]$$
$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']]\right]$$
$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s')\right], \quad \forall s \in \mathcal{S}, \tag{12}$$

- Notice that we have merged the two sums over r and s' in the last expression. This is to ease notation.
- This expression can be read as an expected value. It sums over the triple (a, s', r). For each triple, we compute its probability $\pi(a|s)p(s', r|s, a)$.

Bellman Equation

- Equation 12 is called Bellman equation. We will learn that the Bellman equation is the basis to compute, approximate, and learn v_π.
- We can write similar equations for the action-values:

$$\begin{aligned} q_\pi(a,s) &\doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \sum_{s'}\sum_{r} p(s',r|s,a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \end{aligned} \tag{13}$$

Outline

Optimal Policies and Optimal Value Functions

- Solving an RL task means finding a policy that maximizes reward in the long run.
- In our MDP setting, VFs define a partial ordering over policies.
- A policy π is said to be better than or equal to a policy π' if:

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \quad \forall s \in \mathcal{S}.$$

- There is always at least one policy that is better than or equal to all other policy: *optimal policy*.
- There can be more than one optimal policy. We denote all of them by π_*. These policies share the same state-value function, called the *optimal state-value function*, given by:

$$v_*(s) \doteq \max_\pi v_\pi(s), \quad \forall s \in \mathcal{S}. \tag{14}$$

Optimal Policies and Optimal Value Functions

- Optimal policies also share the same optimal action-value function, denoted by q_*, and:

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a), \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s) \tag{15}$$

- This function gives the expected return for taking action a in state s and thereafter following an optimal policy:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \tag{16}$$

- v_* is the value function for a policy, so it must satisfy Bellman-equation (12). This is called the *Bellman optimality equation*:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi*}(s,a) = \max_a \mathbb{E}_{\pi*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]. \end{aligned} \tag{17}$$

Optimal Policies and Optimal Value Functions

- The Bellman optimality equation for q_* is:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s' r} p(s', r | s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right] \end{aligned} \tag{18}$$

- For finte MDPs, the Bellman equation for v_* has a unique solution. Actually, the Bellman optimality equation is a system of n equations for n states.
- If the dynamics p are known, one can solve the system of equations for v_* using any method for solving systems of nonlinear equations. One can also solve the system for $q*$.
- Once you have v_*, it is relatively easy to determine an optimal policy. Any policy that is *greedy* with respect to v_* is an optimal policy.
- v_* is beautiful because a greedy policy is already optimal given that v_* already takes into account the reward consequence of all possible future behavior.

$q_*(s, a)$ MAKES YOUR LIFE EVEN EASIER

- If you have q_* you don't have to do a one-step-ahead search: for any state s, find any action that maximizes $q_*(s, a)$.
- he action-value function effectively caches the results of all one-step-ahead searches.
- At the cost of representing a state-action pair, instead of just states, the optimal action value function allows optimal actions to be selected without having to know anything about possible successor states and their values (**that is, without having to know anything about the environment's dynamics**).

Bellman Optimality Equations for Recycling Robot

Some Remarks

- Solving the Bellman optimality equation provides a route for optimal policy.
- This solution is rarely directly useful. It is akin to exhaustive search, looking ahead at all possibilities, computing probabilities of occurrence and their desirabilities in terms of expected rewards.
- This way of solving the RL problem relies on three assumptions:
  1. We accurately know the dynamics of the environment.
  2. We have enough computational resources.
  3. The Markov Property.
- For real applications, most of the time, at least one of these assumptions is violated. In RL, one typically has to settle for approximate solutions.
- In the next sessions, we consider a variety of such methods.

Optimality and Approximation

- We have defined optimal value functions and optimal policies.
- Computing the optimal value function rarely happens. For most applications, these computations are extremely costly.
- However, a well-defined notion of optimality organizes the approach to learning and a way to understand the theoretical properties of different algorithms.
- For instance, board games such as chess are a tiny fraction of human experience, yet large, custom-designed computers still cannot compute optimal moves.
- Memory is also an important constraint. In many cases, there are far more states than could be entries in a table.
- Our framing of RL problems and their complexities also presents us with some unique opportunities. The nature of RL makes it possible to approximate optimal policies in ways that put effort into learning to make good decisions in frequently encountered states.

Outline

Conclusions

- RL is about learning from interaction how to behave in order to achieve a goal.
- The RL agent and its environment interact over a sequence of discrete time steps.
- The specification of their interface defines a particular task: *actions*.
- States are the basis for making choices, and rewards are the basis for evaluating the choices.
- Policy is a stochastic rule by which the agent selects actions as a function of states. Goal: maximize the amount of rewards received over time.

Conclusions

- Policy's value functions assign to each state, or state-action pair, the expected return from that state, or state-action pair, given that the agent uses the policy.
- The optimal value functions assign to each state, or state-action pair, the largest expected return achievable by any policy.
- Policies whose value functions are optimal are optimal policies.
- Optimal value function for state and action-state pairs are unique for a given MDP, but there can be multiple optimal policies.
- Bellman optimality equations are special consistency equations that the optimal value functions must satisfy and that in principle can allow to solve the optimal value functions.
- Computing the exact VFs and policies is usually extremely costly; most of the time, we aim to approximate them as best as we can.