Learning in Robotics (ESE 650)
Reinforcement Learning

Pedro A. Ortega

UNIVERSITY of PENNSYLVANIA

April 17, 2014

What is Reinforcement Learning (RL)?

RL is a machine learning technique for learning to control a system from reinforcement signals. This is useful when:

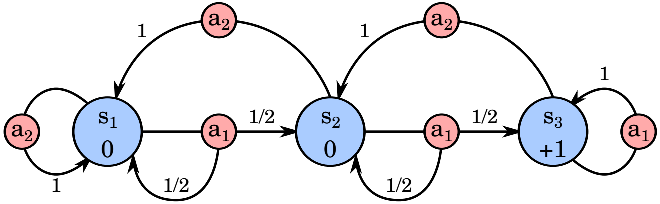
1. we do not know the system dynamics;
2. we do not know the goal beforehand, but we have a teacher.

In practice, we can use RL to learn or refine a policy when we do not have an accurate model of the system due to its complexity, e.g. robot snakes and biped robots.

Markov Decision Processes (MDPs)

In RL, a system is typically conceptualized as an MDP.
An MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where

- $\mathcal{S}$ is a set of states;
- $\mathcal{A}$ is a set of actions;
- P is a transition probability function: $P(s'|s, a)$;
- R is a (bounded, real-valued) reward function: $R(s)$.



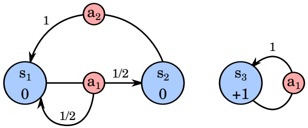
Policies

A controller is formalized as a policy. A policy $\pi : \mathcal{S} \to \mathcal{A}$ maps a state $s \in \mathcal{S}$ into an action $\pi(s) \in \mathcal{A}$.

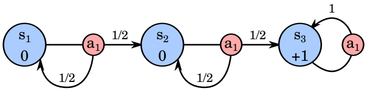
	s_1	s_2	s_3
π_1	a_1	a_2	a_1
π_2	a_1	a_1	a_1

When combined, an MDP and a policy define a Markov chain over states with transition probabilities $P(s'|s, \pi(s))$.

π_1



π_2



Goal

The goal is to choose a policy maximizing a cumulative sum of rewards. Typically, choose π such that

$$\mathbb{E}\left\{ R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots \middle| \pi \right\} = \mathbb{E}\left\{ \sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| \pi \right\}$$

is maximized, where $\gamma \in (0,1)$ is a discounting factor.

Obs: There are other forms of discounting. To be well-defined, the cumulative sum of rewards must converge as $t \to \infty$.

Value Function

The objective function can be characterized recursively. For any policy π and initial state $s_0 \in \mathcal{S}$, define the value

$$V^{\pi}(s_0) = \mathbb{E}\left\{ \sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| \pi \right\}.$$

Then,

$$\begin{aligned} V^{\pi}(s_0) &= \mathbb{E}\left\{ R(s_0) + \gamma \sum_{t=0}^{\infty} \gamma^t R(s_{t+1}) \middle| \pi \right\} \\ &= R(s_0) + \gamma \mathbb{E}\left\{ V^{\pi}(s_{t+1}) \middle| \pi \right\} \\ &= R(s_0) + \gamma \sum_{s_1} P(s_1 | s_0, \pi(s_0)) V^{\pi}(s_1). \end{aligned}$$

Optimal Policy

The value function for policy π is

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s,a) V^{\pi}(s'), \qquad \forall s \in \mathcal{S}.$$

The optimal policy π_* maximizes the value function.

$$V^{\pi_*}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^{\pi_*}(s'), \qquad \forall s \in \mathcal{S}.$$

These are the Bellman optimality equations.

When P and R are known, then the optimal policy can be calculated (up to an arbitrary precision) using dynamic programming.

Learning the Optimal Policy

When we do not know P and R, then we need to learn it from simulations.

1. Brute-force Monte Carlo: Simulate the system under any policy, estimate the value function using many rollouts. Finally, pick the best policy. Highly inefficient.
2. Estimate the optimal value function directly. One way to do this is using Q-Learning.

Q-Learning

Q-Learning is a model-free RL technique that attempts to learn the optimal policy online. It learns a function that measures the quality of state-actions pairs, called the Q-Function:

$$\begin{aligned} Q(s,a) &:= R(s) + \gamma \sum_{s'} P(s'|s,a) V^{\pi^*}(s') \\ &= R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a'). \end{aligned}$$

This is useful, because

$$\pi^*(s) = \arg\max_a Q(s,a).$$

We can estimate the Q-Function using Monte Carlo techniques.

Method 1: Empirical Averages

Estimates the expectation of $f(x)$ using samples $f^{(1)}(x), \ldots, f^{(N)}(x)$

$$F(x) = \mathbb{E}\{f(x)\} \approx \frac{1}{N} \sum_{n=1}^{N} f^{(n)}(x)$$

Unfortunately, the calculation of empirical averages is not straightforward due to the recursive definition of the Q-Function:

$$Q(s,a) \approx \frac{1}{N} \sum_{n=1}^{N} \left\{ R^{(n)}(s) + \max_{a} Q^{(n)}(s', a') \right\}$$

The Q-Values cannot be directly observed, because they depend on the future realization.

Method 2: Exponential Smoothing

Keeps a running estimate $F_t(x)$ of $\mathbb{E}\{f(X)\}$ and updates online:

$$F_{t+1}(x) = (1-\alpha)F_t + \alpha f^{(t)}(x),$$

where $\alpha \in (0,1)$. This yields the classical Q-Learning update rule:

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha\Big\{R(s_t) + \max_{a'} Q_t(s',a')\Big\}.$$

Observations:

1. This learning rule uses bootstrapping!
2. Updates only after seeing the next state s'.

Method 3: Stochastic Gradient Descent

Similar to exponential smoothing, but uses a parameterized family $F_\theta(x)$, where $\theta \in \Theta$, and updates the parameter as

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial E_t}{\partial \theta}\bigg|_{\theta_t}, \qquad E_t = (F_{\theta_t}(x) - f^{(t)}(x))^2,$$

where $\alpha \in (0,1)$. We use $Q_\theta(s,a)$:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \alpha \frac{\partial E_t}{\partial \theta} = \theta_t - \alpha \frac{\partial E_t}{\partial Q(s,a)} \frac{\partial Q(s,a)}{\partial \theta} \\ &= \theta_t - 2\alpha \Big(Q_{\theta_t}(s,a) - R(s) - \max_{a'} Q_{\theta_t}(s',a') \Big) \cdot \frac{\partial Q_\theta(s,a)}{\partial \theta}\bigg|_{\theta_t}\end{aligned}$$

Modelling the Q-Function

- Discrete state-action spaces: Based on a Q-table. Requires discretization: e.g.
  1. clustering
  2. kernel estimator

  Learning can be very slow.
- Continuous state-action spaces: Requires choosing smooth model class:
  1. radial basis functions
  2. feedforward neural network

Convergence of Estimator

- Recursive definition $\Rightarrow$ Q-Function is learned through relaxation. Typically, $\alpha_t \to 0$:

$$\sum_t \alpha_t = \infty, \qquad \sum_t \alpha_t^2 < \infty.$$

- Since we only “observe” the Q-Function locally, convergence requires sufficient mixing: every state must be visited infinitely often.
- Ergodicity: every state must always be reachable.
- Exploration-exploitation trade-off.

Ergodicity

Ergodic MDP: For every $s, s' \in \mathcal{S}$, there exists a policy $\pi : \mathcal{S} \to \mathcal{A}$ such that

$$P(s'|s,\pi) > 0,$$

where $P(s'|s,\pi)$ is the probability of reaching state s' from state s under policy π.

Unstable systems are non-ergodic. They can be made ergodic with episodes.

Exploration-Exploitation

How do we choose actions online? We have seen that

$$\pi^*(s) = \arg\max_a Q(s, a).$$

However, at time t we only have an estimate of $Q(s, a)$. Commiting too early to a learned policy is sub-optimal!

The easiest algorithm is ϵ-greedy:

- Follow the greedily optimal policy with probability $(1 - \epsilon)$,
- otherwise choose a random action.
- Typically, we let $\epsilon_t \to 0$.

Advanced Exploration-Exploitation

We can also use optimism in the face of uncertainty.

The most famous algorithm of this class is UCB1. Define

$$UCB_t(s,a) := Q_t(s,a) + \sqrt{\frac{2\log t}{N(s,a)}}.$$

Then, the UCB1 policy is

$$\pi_{\mathrm{UCB1}}(s) = \arg\max_a \{UCB_t(s,a)\}.$$

Obs: This requires keeping track of $N(s,a)$!

Conclusions

1. Systems can be formalized as Markov decision processes.
2. The optimal policy is characterized with the value function.
3. When transitions are unknown, we can learn the policy using reinforcement learning.
4. Q-Learning learns the state-action value of a state online.
5. A good policy must balance exploration and exploitation.