MS&E338 Reinforcement Learning Lecture 1 - April 2, 2018

Introduction

Lecturer: Ben Van Roy *Scribe: Gabriel Maher*

1 Reinforcement Learning Introduction

In reinforcement learning (RL) we consider an **agent** interacting with an **environment**. The agent selects **actions** and, based on the selected action, observes **states** and **rewards** from the environment. The RL problem is then to design an agent that selects **optimal actions** in some sense. Typically optimality is defined here as maximizing some form of **expected long-term reward**. An algorithm/heuristic for finding an optimal agent is a possible **solution**, or **RL algorithm**, to the RL problem.

There are many RL algorithms and specific instances of RL problems that they are tested on. A major criteria for an RL algorithm is the amount of time it takes to find the optimal agent, known as **data-efficiency**. RL problems are typically organized into a series of **episodes**, where in an episode the environment is set to an initial state and the agent is allowed to interact with the environment, until some specified stopping criteria is met. Episodes are then repeated until we run out of computation time, or a solution is found. Data-efficient algorithms thus require fewer episodes than less efficient algorithms.

As an example, consider the cart-pole problem (see e.g. `https://gym.openai.com/envs/CartPole-v1/`), where the environment consists of a robot balancing a pole on top of itself. The state is the position and velocities of the robot and pole. The agent can move the robot left or right to try to balance the pole (these are the actions). At every step the agent receives a higher reward the closer to balanced the pole is.

Cart-pole is an easy RL problem, in that most algorithms do not need many episodes to find an optimal agent. However, if the problem is modified such that the pole starts at the bottom and the agent only receives a reward if the pole is very close to balanced (no reward otherwise), algorithms may take millions of episodes to find a solution and are thus not very data-efficient. The fact that it is easy to construct simple RL problems, that are difficult for many algorithms creates suspicion that many RL algorithms are brittle. There is therefore possibly still a lot of work to be done to create truly robust RL algorithms.

2 Markov Decision Processes

To know what we are talking about, we need a precise formulation of the RL problem. Typically RL problems are formulated using concepts from **Markov Decision Processes (MDPs)**. With MDPs we define the states, actions, rewards and environment as as a tuple $(\mathcal{S}, \mathcal{A}, R, P, \rho)$ where:

- $\mathcal{S}$ - Set of possible states
- $s_t \in \mathcal{S}$ - state at step t.
- $\mathcal{A}$ - Set of possible actions
- $a_t \in \mathcal{A}$ - selected action at step t.
- R - Reward function. The reward at step t is given by $r_{t+1} = R(s_t, a_t, s_{t+1})$.
- P - transition probabilities such that $s_{t+1} \sim P(s|s_t, a_t)$, i.e. the next observed state depends on the current state and selected action.
- ρ - Initial state distribution such that $s_0 \sim \rho(s)$.

The MDP gives us a precise formulation of the environment, given a state s_t we select an action a_t and observe s_{t+1} and r_t according to the transition probabilities P.

How do we define an agent? This can be done using a policy function $\pi(a|s)$. A policy function maps from states to actions and can be either deterministic or non-deterministic (random). In the non-deterministic case we have $a_t \sim \pi(a|s_t)$, i.e. we sample the current action based on the current observed state.

Given an MDP and a policy, an episode thus consists of repeating the following loop (given the initial state $s_0 \sim \rho(s)$):

1. $a_t \sim \pi(a|s_t)$
2. $s_{t+1} \sim P(s|s_t, a_t)$
3. $r_{t+1} = r(s_t, a_t, s_{t+1})$

which produces the observed states, actions and rewards:

$$\text{episode} := s_0, a_0, r_1, s_1, a_1, r_1, ..., s_{\tau-1}, a_{\tau-1}, r_{\tau-1}, s_\tau \tag{1}$$

where τ is the **stopping time**.

Since the transition probabilities P are probabilities, we have:

$$\sum_{s' \in \mathcal{S}} P(s'|s, a) \leq 1 \tag{2}$$

That is for any current state, s, and action, a, the probabilities of moving to any of the next states must sum to less than 1, or sum to 1. The "less than" comes from the fact that there may also be a non-zero probability of terminating the episode.

2.1 Assumptions

Note that in the lecture we made a few key assumptions to simplify things. In general the state and action spaces, $\mathcal{S}$ and $\mathcal{A}$, can be uncountably infinite (e.g. a continuous interval). This makes analysis complicated, so here we assume that both the state and action space are finite sets.

Furthermore the stopping time τ can be infinite, but here we assume that $\tau < \infty$ almost surely.

2.2 Key differences between RL and MDPs:

There are a few differences between RL problems and MDPs that are important to note. With an RL problem, the transition matrix P is unknown and must either be learned by interacting with the environment, or algorithms that do not require explicit knowledge of P must be used. In an MDP, P is known and hence can be used to find the optimal agent. Thus RL problems are more general and harder than MDPs. However, it is useful to first study MDPs as they do share many similarities with RL problems.

3 Optimal Solutions to Markov Decision Process Problems

We now know what an MDP is and how to define the environment and agent. But how do we actually find an optimal agent? What does optimal even mean in this context? With the MDP formulation we see that, given the policy, transition probabilities and initial state distribution, the actions, states and rewards follow a random process. We can thus try to find a policy that maximizes the expected reward:

$$\max_\pi E\left[\sum_{t=1}^{\tau} r_t\right] \tag{3}$$

where the expectation is over the observed states and rewards given that actions are selected from the policy π.

The specific initial state influences the observed trajectory of the episode, fixing the initial state we can define the **value function**:

$$V^{\pi}(s) = E_{\pi}\left[\sum_{t=1}^{\tau} r_t | s_0 = s\right] \tag{4}$$

which is the expected long-term reward starting from state s and then following actions given by π. Considering that the best policy maximizes the expected long-term reward we can then define the optimal value function:

$$V^{*}(s) = \max_{\pi} E_{\pi}\left[\sum_{t=1}^{\tau} r_t | s_0 = s\right] \tag{5}$$

i.e. $V^*(s)$ is the expected long-term reward, starting from state s and then following actions from the optimal policy. It is important to note that there is an expected long-term reward value for each state $s \in \mathcal{S}$, therefore V is a vector, or function.

We can set up a recursive relation for the Value functions, known as **Bellman's equation**. The intuition is that the current expected long-term reward can be thought of as the expected next reward plus the expected long-term reward from the next state. Concretely:

$$V^{\pi}(s) = E\left[r_{t+1} + V^{\pi}(s_{t+1}) | s_t = s, a_t \sim \pi(a|s)\right] \tag{6}$$

and

$$V^{*}(s) = \max_{a \in \mathcal{A}} E\left[r_{t+1} + V^{*}(s_{t+1}) | s_t = s, a_t = a\right] \tag{7}$$

the particular action a that attains $V^*(s)$ is the optimal action in state s, denoted as a^*.

3.1 Bellman Operators

Suppose that we are at state s_t and want to know the value of this state, $V(s_t)$. From the Bellman equations we see that if we know the value in all possible next states, $V(s_{t+1})$ we can calculate $V(s)$. We can think of this procedure as an operator that maps the value function of the next states to the value function of the current state. This is the intuition for the **Bellman Operator**:

$$(T_{\pi}V)(s) = E\left[r_{t+1} + V^{\pi}(s_{t+1}) | s_t = s, a_t \sim \pi(a|s)\right] \tag{8}$$

$$(TV)(s) = \max_{a \in \mathcal{A}} E\left[r_{t+1} + V^{*}(s_{t+1}) | s_t = s, a_t = a\right] \tag{9}$$

The Bellman operator notation will be useful when we define algorithms for solving RL problems.

4 Solving Markov Decision Process Problems

How do we actually find the optimal value function or policy? Intuitively the Bellman equations tell us the relation between the value function at different steps. It turns out that if we start with an initial guess for the value of every state, and repeatedly solve the Bellman equation, we find the optimal value function:

Algorithm 1 Value-iteration

```
Initialize V_0
for k = 0, 1, 2, ... do
    V_{k+1} ← T V_k
end for
```

What about finding the policy? Similarly, given a policy, we can use the Bellman equations to calculate the values and, given values, we can find a better policy. Iterating this process we can find the best policy:

Algorithm 2 Policy-iteration

Initialize policy π_0
for $k = 0, 1, 2, ...$ **do**
  solve $V_k = T_{\pi_k} V_k$ for V_k
  solve $T_{\pi_{k+1}} V_k = T V_k$ for π_{k+1}
end for

4.1 Linear programming

Let

$$\overline{R}(s,a) := E\left[r_t | s_t = s, a_t = a\right] \tag{10}$$

be the expected reward when being in a particular state and selecting a particular action. We can solve the MDP by finding the expected frequency counts, $\mu(s,a)$, between actions and states that maximizes the expected reward for all states and actions. This is a constrained optimization problem:

$$\begin{aligned} &\max_{\mu} \sum_{s\in\mathcal{S}, a\in\mathcal{A}} \overline{R}(s,a)\mu(s,a) \\ &\text{s.t. } \mu(s,a) \geq 0 \quad \forall s,a \\ &\qquad \rho(s') + \sum_{s,a} \mu(s,a) P(s'|s,a) = \sum_{a} \mu(s',a) \quad \forall s' \end{aligned} \tag{11}$$

where the final constraint is needed to ensure that $\mu(s,a)$ is consistent.

4.2 Dual Formulation

We can also use a dual formulation that lets us solve for the value function:

$$\begin{aligned} &\min_{V} \sum_{s\in\mathcal{S}} \rho(s) V(s) \\ &\text{s.t. } V(s) \geq (TV)(s) \ \forall\ s \end{aligned} \tag{12}$$

The constraint is nonlinear (max in the Bellman operator) and therefore difficult to work with. We can convert it to a system of linear constraints by noting that if $V(s) \geq TV(s)$ for the optimal action a^*, $V(s)$ will also be greater for all other possible actions, this leads to:

$$V(s) \geq \overline{R}(s,a) + \sum_{s'\in\mathcal{S}} P(s'|s,a) V(s') \quad \forall\ s,a \tag{13}$$

5 Properties of policies and value functions

To prove that value iteration and policy iteration converge to optimal solutions, we need to establish some properties of the Bellman operator.

5.1 Monotonicity

$$\forall\ \pi \text{ if } V(s) \leq V'(s)\ \forall\ s \in \mathcal{S} \text{ then } (T_\pi V)(s) \leq (T_\pi V')(s)\ \forall\ s \in \mathcal{S} \tag{14}$$

proof:

$$\begin{aligned} (T_\pi V)(s) &= E\left[r_{t+1} + V(s_{t+1}) | s_t = s, a_t \sim \pi(a, s_t)\right] \\ &\leq E\left[r_{t+1} + V'(s_{t+1}) | s_t = s, a_t \sim \pi(a, s_t)\right] \\ &= (T_\pi V')(s) \end{aligned} \tag{15}$$

$$\text{if } V(s) \leq V'(s) \ \forall \ s \in \mathcal{S} \text{ then } (TV)(s) \leq (TV')(s) \ \forall \ s \in \mathcal{S} \tag{16}$$

proof:

$$\begin{aligned}(TV)(s) &= \max_a E\left[r_{t+1} + V(s_{t+1})|s_t = s, a_t = a\right] \\ &\leq \max_a E\left[r_{t+1} + V'(s_{t+1})|s_t = s, a_t = a\right] \\ &= (TV')(s)\end{aligned} \tag{17}$$

5.2 Contraction mapping

$$||TV - TV'||_{\infty,1/\tau} \leq \alpha||V - V'||_{\infty,1/\tau} \tag{18}$$

where

$$\tau(s) = \max_\pi E_\pi\left[\tau|s_0 = s\right] < \infty \tag{19}$$

is the maximum stopping time when starting from s and

$$||V||_{\infty,1/\tau} = \max_{s\in S} \frac{|V(s)|}{\tau(s)} \tag{20}$$

We then have

$$\alpha = \max_s \frac{\tau(s) - 1}{\tau(s)} < 1 \tag{21}$$

Thus the Bellman operator T is a contraction mapping under the specified norm. **proof:** See lecture 2.