DDA4230 Reinforcement learning Optimality of MDPs

Lecture 2

Lecturer: Guiliang Liu *Scribe: Baoxiang Wang*

1 Goal of this lecture

In this lecture we will introduce the optimality of Markov decision processes and discuss some examples of MDPs.

Suggested reading: Chapter 3 and 4 of *Reinforcement learning: An introduction*; *Human-level control through deep reinforcement learning* (Volodymyr Mnih et al. Nature 2015); Chapter 1 of *Reinforcement learning: Theory and algorithms*.

2 Recap: Markov decision processes

We consider the discrete-time Markov decision process (MDP) setting, denoted as the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \rho_0, \gamma)$.

- $\mathcal{S}$ the state space;
- $\mathcal{A}$ the action space. $\mathcal{A}$ can depend on the state s for $s \in \mathcal{S}$;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ the environment transition probability function;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathbb{R})$ the reward function;
- $\rho_0 \in \Delta(\mathcal{S})$ the initial state distribution;
- $\gamma \in [0, 1]$ the unnormalized discount factor.

Note that $\Delta(\mathcal{X})$ denotes the set of all distributions over set $\mathcal{X}$.

A stationary MDP follows for $t = 0, 1, \dots$ as below, starting with $s_0 \sim \rho_0$.

- The agent observes the current state s_t;
- The agent chooses an action $a_t \sim \pi(a_t \mid s_t)$;
- The agent receives the reward $r_t \sim \mathcal{R}(s_t, a_t)$;
- The environment transitions to a subsequent state according to the Markovian dynamics $s_{t+1} \sim \mathcal{T}(s_t, a_t)$.

This process generates the sequence $s_0, a_0, r_0, s_1, \dots$ indefinitely. The sequence up to time t is defined as the trajectory indexed by t, as $\tau_t = (s_0, a_0, r_0, s_1, \dots, r_t)$.

The goal is to optimize the expected return

$$\mathbb{E}_{s_t, a_t, r_t, t \geq 0} [R_0] = \mathbb{E}_{s_t, a_t, r_t, t \geq 0} \big[\sum_{t=0}^{\infty} \gamma^t r_t \big]$$

over the agent's policy π.

3 Optimality of MDPs

We now have all the background necessary to discuss the problem of reinforcement learning, where we seek to find a best policy that achieves the greatest value function among the set of all possible policies. To get started, we need to address the question

what do we exactly mean by finding an optimal policy?

We first define precisely what it means for a policy, not necessarily stationary, to be an optimal policy. The existence of an optimal policy will be assumed throughout the course unless otherwise mentioned.

Definition 1 *A policy π^* is an optimal policy if for every policy π, for all states $s \in S$, $V^{\pi^*}(s) \geq V^{\pi}(s)$.*

When the MDP is non-stationary or is with a finite horizon, the definition of optimality will be $V_t^{\pi^*}(s) \geq V_t^{\pi}(s)$ for every π, s, and t.

In the setting of stationary, infinite-horizon MDPs (which is the MDP defined in this lecture notes), if some not-necessarily stationary policy is optimal, then at least one stationary policy is optimal. An intuitive proof sketch is constructive. Simply specify a not-necessarily stationary optimal policy and drop its dependency on t (for example, by setting $t = 0$). This will obtain a stationary optimal policy.

For stationary MDPs, if some not-necessarily deterministic policy is optimal, then at least one deterministic policy is optimal. The proof is also constructive. We can first specify a not-necessarily deterministic optimal policy and then define a new deterministic policy, whose action is an arbitrarily action of the optimal policy with probability greater than 0.

Taking discrete state space $\mathcal{S} = [n]$ and action space $\mathcal{A} = [m]$ as an example, the total number of policies is m^n. On the contrary, this number is m^{nT} if we allow non-stationarity for some horizon T. For stochastic policies, the size of the universe will be infinity. We see how significant stationary and deterministic policies reduce the size of the universe of policies when searching for an optimal policy. Note that when the action space is depending on the state, this number of policies is $\prod_{s \in \mathcal{S}} |\mathcal{A}(s)|$.

We have thus established the existence of an optimal policy and moreover concluded that a deterministic stationary policy suffices. This then allows us to make the following definition.

Definition 2 *The optimal state value function for an infinite horizon MDP is defined as*

$$V^*(s) = \max_{\pi \in \Pi} V^{\pi}(s) \ , \tag{1}$$

and there exists a stationary deterministic policy $\pi^* \in \Pi$, which is an optimal policy, such that $V^*(s) = V^{\pi^*}(s)$ for all states $s \in \mathcal{S}$, where Π is the set of all stationary deterministic policies.

The optimal value function is unique for an MDP. Otherwise if two value functions do not agree on some state, the one possessing the smaller value is not the optimal value function. The uniqueness of optimal policies does not hold in general.

3.1 Dynamic programming

For those who have learned algorithms and data structures before, the problem of sequential decision making can be solved via dynamic programming for the finite horizon case. Denoting $V_t^*(s)$ to be the optimal value function at time t,

$$V_T^*(s) = 0\,,$$
$$V_t^*(s) = \max_a \ \mathbb{E}[\mathcal{R}(s,a)] + \gamma \sum_{s' \in S} \mathbb{P}(s' \mid s,a) V_{t+1}^*(s')\,, \quad \forall\, t = 0, \ldots, T-1\,.$$

These equations immediately lend themselves to a dynamic programming algorithm. However, this requires the knowledge of the world model ($\mathcal{R}(s,a)$ and $\mathbb{P}(s' \mid s,a)$), which is then out of the scale of this course. The optimality of MDPs described by dynamic programming can be extended to a variety of algorithmic approaches, like value iteration and Q-learning, which will be discussed later in our lectures.

3.2 The Bellman optimality equation

The Bellman optimality equation, named after Richard E. Bellman, is a necessary condition for a value function to be optimal. It describes the recursive property of the optimal value function

$$V^*(s_t) = \max_a \ \mathbb{E}\left[r_t + \gamma V^*(s_{t+1}) \mid a_t = a\right].$$

It is worth noting that the Bellman optimality equation is different from the Bellman equation. The Bellman equation describes an arbitrary policy's value function and it therefore averages over a_t instead of taking the maximum, which writes $V(s_t) = \mathbb{E}[r_t + \gamma V(s_{t+1})]$.

This property is critical and it leads to many algorithms to learn the optimal policy and the optimal value.

Value iteration LHS is no larger than RHS if we replace V^* by a not-necessarily optimal value function V. Therefore for discrete state and action spaces, we can assign RHS to V and repeat the iteration. This leads to improvements of the current value for each iteration and V will converge to the optimal value function under some conditions.

Q-learning Despite that the property is necessary but not sufficient, a set of algorithms seek function who fulfill this property and are then more likely to be close to optimal. The approach to satisfy the Bellman optimality equation is to define a Bellman error (e.g. $\frac{1}{2}(LHS - RHS)^2$) and minimize it. It is commonly presented on an alternative form of $Q^*(s_t, a_t) = \max_a \mathbb{E}[r_t + \gamma Q^*(s_{t+1}, a)]$, where Q^* is the action value function for an optimal policy, known as the optimal action value function.

4 Examples of MDPs

4.1 Boyan chains

The Boyan chain for $N+1$ states is a Markov chain with the transition probability shown in Figure 1. The starting state is 1. It has a reward of -1 for every step except for the

terminal state and the process terminates at state $N+1$. The terminal state has no reward.

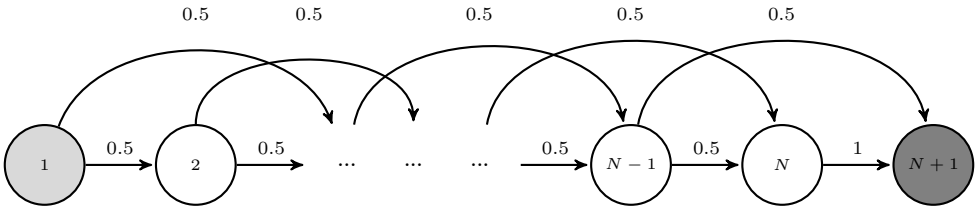

Figure 1: The Boyan Markov chain.

The value function $V(s)$ is then the negative number of steps elapsed starting from state s until the terminal state. Then by the linearity of expectation, we have $V(s) = \frac{1}{2}V(s+1) + \frac{1}{2}V(s+2) - 1$ for $s \leq N-1$. With the boundary conditions $V(N) = -1$ and $V(N+1) = 0$ we find that

$$V(s) = \frac{4}{9} - \frac{2}{3}(n - s + 2) - \frac{4}{9}(-\frac{1}{2})^{n-s+2}\,.$$

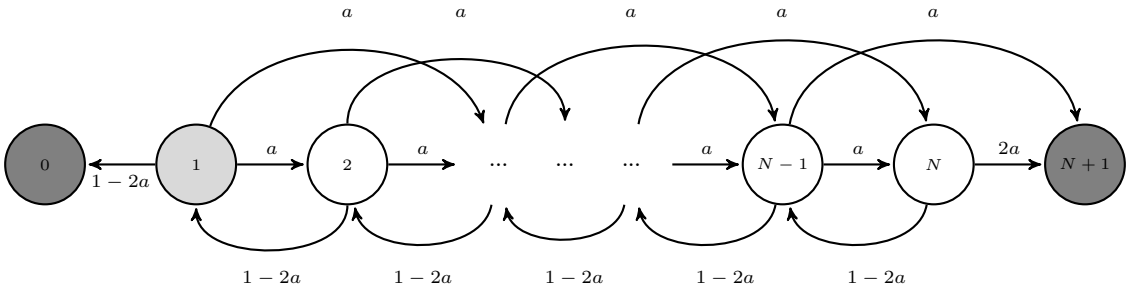

Figure 2: The Boyan chain variant. With the defined action and reward, this example demonstrates the importance of exploration in reinforcement learning.

Some variants of Boyan chain are classical games to test the ability of agents to explore in an MDP and to not stuck at local optimums. Figure 2 gives an example. An action $a \in [0, 0.5]$ needs to be decided by the agent at every state and the transition probability is shown according to the figure. The starting state is 1 and the set of terminal states is $\{0, N+1\}$. The reward is -1 for every transition shown in the figure, and the reward at state 0 and state $N+1$ (despite the action in the last step) are 1 and N^2, respectively.

It is hard, in this task, to find the optimal policy which chooses $a = 0.5$ at every state. It is relatively easy for the learning algorithm to find state 0 which gives a positive reward while terminates the sequence of -1 rewards. By setting $a = 0$ in the starting state, the algorithm can generate a return of $-1+\gamma$, which is better than most of the policies. It can be very hard for an agent to ever explore the state $N+1$, for large N, which is the bare necessity to learn the optimal policy.

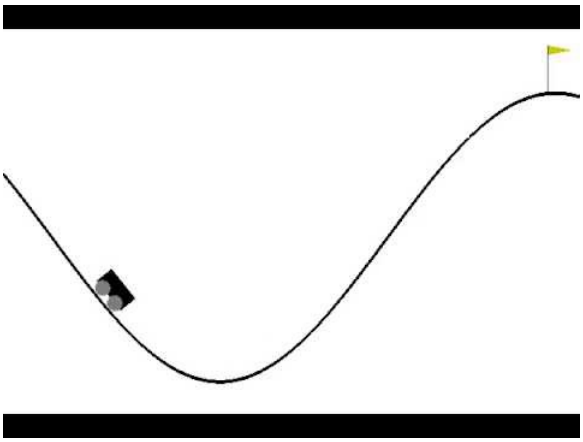

Figure 3: The Mountain Car problem.

4.2 Mountain Car

The problem describes the decision-making of a car driver who, starts from a valley, aims to drive to the mountain peak (the flag). It is however insufficient to drive directly from the valley rightwards.

As it requires the agent to move left, which initially seems to be moving further away from the goal, it is a classical demonstration of delayed gratification and the problem of temporal credit assignment.

4.3 Montezuma's Revenge

Montezuma's Revenge is one of the Atari 2600 games, which compose the Atari learning environment (ALE) commonly used in reinforcement learning tests. In this game, the agent needs to complete specific series of actions over extended periods of time. In the case of the first room of Montezuma's Revenge (see Figure 4), this means descending a ladder, jumping across an open space using a rope, descending another ladder, jumping over a moving enemy, and then finally climbing another ladder. All of these get the agent the very first key in the very first room. In the first level there are 23 more such rooms for the agent to navigate through in order to complete the level. The failure conditions in the game are fairly strict, with the agent's death happening due to any number of possible events, the most punishing of which is simply falling from too high a place.

Unlike the vast majority of the games in the ALE, which are solved by deep Q-learning in its early variants, Montezuma's Revenge has been a challenge for deep RL for long. What distinguishes Montezuma's Revenge from other games in the ALE is its sparse rewards and its long horizon. These induce both problems in exploration and problems in credit assignment. On the former, the agent has to ever touch some positive reward to receive a reinforcement signal. On the latter, even if the agent does, it has to properly credit the reinforce to its long sequence of actions that eventually lead to the signal. Due to this notorious difficulty, the game has been seen as a kind of grand challenge for deep RL methods.

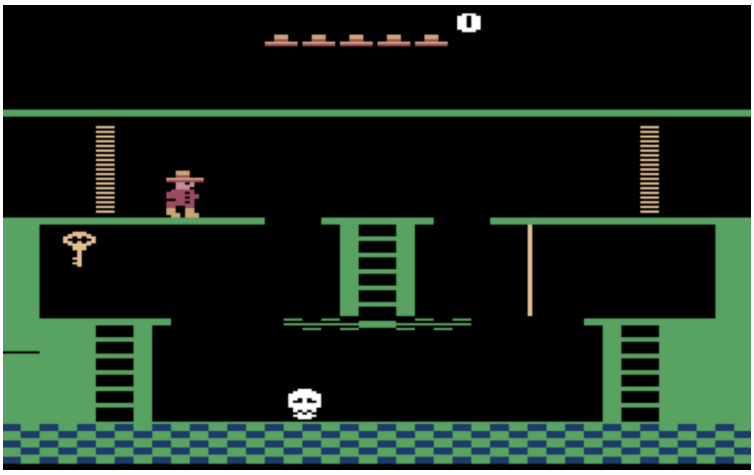

Figure 4: Montezuma's Revenge.

Acknowledgement

This lecture notes partially use material from *Reinforcement learning: An introduction* and *CS234: Reinforcement learning* from Stanford. Figure 1 and 2 are adapted from *Preconditioned temporal difference learning* by Hengshuai Yao and Zhi-Qiang Liu and are drawn by Shaokui Wei. The description of Montezuma's Revenge partially uses the blogpost *On "solving" Montezuma's Revenge* on Medium by Arthur Juliani. Proofread by Shaokui Wei.