CSE6243: Advanced Machine Learning **Fall 2024**

Lecture 20: MDP - Bellman Recursion

Lecturer: Bo Dai *Scribes: Brieuc Popper, Kausar Patherya*

Note: *LaTeX template courtesy of UC Berkeley EECS Department.*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

1.1 Recap

There was no recap for this lecture as we moved on to a whole new topic.

1.2 Reinforcement Learning vs. Other Learning

Reinforcement Learning is a whole new paradigm, that is very different from anything that we saw in class so far, namely

- Supervised Learning
- Unsupervised Learning
- Self-supervised Learning
- Semi-supervised Learning

Let's compare Supervised Learning (SL) and Reinforcement Learning (RL). In SL, you have a dataset D from which are drawn samples $x_i \in D$, and for each sample you have a label y_i.

The goal in SL, given a loss function, is to learn a function f that minimizes $\mathbb{E}_D[\text{loss}(f(x_i), y_i)]$.

In RL, the goal is to learn a policy π such that given the current state $s \in S$, $\pi(s)$ is a distribution over the action space of the next action to take in state s. We are looking for a policy that will maximize the expected cumulative rewards. Typically, we solve something like:

$$\arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right],$$

where γ is the discount factor, r_t is the reward at time step t, and $v(\pi)$ is the expected cumulative reward (value) of the policy π.

In supervised learning, the data distribution is fixed from the start; the only goal is to perform function approximation. In RL, however, depending on the policy which dictates how the agent interacts with the environment, the agent will receive different data. This is an **essential** difference between RL and other types of learning: in RL, the data collection is dynamic and depends on the policy. Additionally, the policy must be optimized to maximize the rewards given by the environment.

1.2.1 Exploration v. Exploitation : a RL-characterizing tradeoff

This leads to a very fundamental tradeoff in RL : the *exploration v. exploitation* tradeoff. The policy needs to be diverse enough so that a lot of diverse data is collected, which is required for the agent to learn how it should behave in all situations it could encounter. Having a diverse policy is the "exploration" part of the tradeoff. You also want to learn a policy that will bring in a lot of reward, and this means that after some point you will have to exploit what you already know about the environment, and start taking actions that you think lead to higher reward, which might restrict your exploration. It is essential for good RL algorithms to find a good balance between exploration and exploitation.

One very simple strategy is to begin with a phase of "pure exploration" where the agent will just sample random actions and observe what reward it gets, and what state it ends up in. Then comes a phase of "pure exploitation" where the agent will just do what it thinks will lead to highest reward based on what it saw in the exploration phase. This could work a little bit, but in practice more sophisticated approaches are necessary. A famous but simple family of approaches to deal with this issue are : $\epsilon - greedy$ approaches. In $\epsilon - greedy$ approaches, a variable ϵ is initialized at starting value, often times 1, and then decreases as the agent's training progresses, to a value like 0.01. At each timestep, with probability ϵ the agent will take a random action, and with probability $1 - \epsilon$ the agent will take a greedy action based on what it currently believes is best.

1.2.2 A common misconception

There is a common misconception that RL is *sample inefficient.* However, this isn't directly true. The reality is that very often RL problems are intrinsically difficult, and it is difficult to solve them without a lot of samples. It is recommended to stay away from RL if there is any other way of solving a given problem. For example, if there is an easy way to apply a well-known SL solution to a problem, one should **not** attempt to solve it with RL. Converting a problem into RL that does not require RL initially will probably increase the difficulty of solving said problem.

1.3 Sequential Decision Making

For finite time horizons, the goal is to maximize the total reward over H time steps:

$$\sum_{t=0}^{H} r_t.$$

For infinite time horizons, we instead maximize the discounted cumulative reward:

$$\sum_{t=0}^{\infty} \gamma^t r_t,$$

where γ is the discount factor, ensuring convergence of the infinite sum.

1.4 Markov Decision Process

This is only one way to formalize the very general problem that RL is. However, it is very frequently used. Markov Decision Process (MDP) μ is defined as a tuple:

$$\mu := \langle S, A, R, P, \gamma, \mu_0 \rangle,$$

where:

- S is the state space.
- A is the action space.
- $R(s,a) : S \times A \to [0, R_{\max})$ is the reward function. Please note that you can make rewards negative as well based on the problem being considered.
- $P(\cdot|s,a) : S \times A \to \Delta(S)$ is the transition probability.
- $\gamma \in [0,1)$ is the discount factor. Note: for problems with finite time horizon, we can instead consider H which is the max number of timesteps, and replace the discounted sum of rewards with a finite sum of rewards. In some specific cases, a discount factor of 1 can be used for undiscounted returns in episodic tasks with a finite horizon.
- $\mu_0 : S \to \Delta(S)$ is the initial state distribution.

To solve a RL problem formalized as an MDP process, the goal is to find a policy π such that given the current state $s \in S$, $\pi(s)$ is a distribution over the action space. Of course, we don't want to find any policy, but rather one that gets the highest possible expected discounted sum of rewards. The objective $J(\pi)$ can be written as:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right].$$

1.5 Value Functions

1.5.1 Value Functions

The value function of a policy π is defined as:

$$v(\pi) := \mathbb{E}_{\mu_0, P, \pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right],$$

and for a finite state space, $V^\pi(s)$ can be represented as a vector:

$$V^\pi := \begin{bmatrix} V^\pi(s_0) \\ V^\pi(s_1) \\ \vdots \\ V^\pi(s_{|S|}) \end{bmatrix}.$$

The state-action value function for policy π is:

$$Q^\pi(s,a) := \mathbb{E}_{P,\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a \right].$$

1.5.2 Optimal Value Functions

The optimal policy π^* maximizes $v(\pi)$. The optimal value functions are:

$$V^*(s) := V^{\pi^*}(s), \quad Q^*(s,a) := Q^{\pi^*}(s,a).$$

1.6 Bellman Equations

1.6.1 Bellman Equation

The Bellman equation for $Q^\pi(s,a)$ is:

$$Q^\pi(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V^\pi(s')].$$

Proof: Consider the expected sum of rewards for a policy π:

$$Q^\pi(s,a) = \mathbb{E}_{P,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right].$$

Breaking down this sum, we take the $t = 0$ term out, which is $r_0 = R(s,a)$, and continue with the sum from $t = 1$:

$$= R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[V^\pi(s')\right].$$

1.6.2 Bellman Optimal Equation

The Bellman optimal equation for V^* is:

$$V^*(s) = \max_{a \in A}\left\{R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V^*(s')]\right\}.$$

Proof:

To derive the Bellman optimality equation, we start by defining the optimal value function, $V^*(s)$, as the maximum expected cumulative reward an agent can obtain starting from state s and following the best possible policy π^*. This means:

$$V^*(s) = \max_\pi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,|\, s_0 = s, \pi\right],$$

where $R(s_t, a_t)$ is the reward obtained at time step t, and $\gamma \in [0,1)$ is the discount factor.

(i) Expressing the Value in Terms of the First Action

Thus, we can rewrite $V^*(s)$ as the maximum reward we can achieve by choosing the best action $a \in A$ and considering both the immediate reward and the discounted value of the optimal policy from the next state s'. This gives:

$$V^*(s) = \max_{a \in A} \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[R(s,a) + \gamma V^*(s')\right].$$

(ii) Expanding the Expectation

Expanding the expectation, we get:

$$V^*(s) = \max_{a \in A}\left\{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')\right\}.$$

Here, $\sum_{s'} P(s'|s,a)V^*(s')$ represents the expected value of following the optimal policy from the next state s'.

(iii) Connection to the Optimal State-Action Value Function Q^*

Define the optimal state-action value function $Q^*(s,a)$ as the maximum expected cumulative reward starting from state s, taking action a, and following the optimal policy thereafter:

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V^*(s')].$$

Then, the relationship between $V^*(s)$ and $Q^*(s,a)$ is:

$$V^*(s) = \max_{a \in A} Q^*(s,a).$$

(iv) Substituting $Q^*(s,a)$ Back into the Equation

Now, substitute the expression for $Q^*(s,a)$ into the equation for $V^*(s)$:

$$V^*(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V^*(s')] \right\}.$$

Conclusion:

This completes the proof that the optimal value function $V^*(s)$ satisfies the Bellman optimal equation:

$$V^*(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V^*(s')] \right\}.$$

This equation captures the recursive nature of optimality in reinforcement learning, where the optimal value of a state is determined by considering the immediate reward plus the discounted value of the next optimal state.

1.7 Problems in Reinforcement Learning

- **Policy Evaluation (Prediction)**: Given a policy π, compute V^π or Q^π or J^π
- **Policy Optimization (Control)**: Find π^* and the corresponding value function (or Q-function)