Reinforcement Learning

Notes

Jingye Wang

wangjy5@shanghaitech.edu.cn

Spring 2020

Contents

5 Markov Decision Process

The large part of this section was done with references [4, 9, 10, 1, 8].

5.1 Markov Reward Process

Markov Reward Process (MRP): *A Markov Reward Process is a tuple* $\langle \mathcal{S}, \mathcal{P}, R, \gamma \rangle$ *where*

- $\mathcal{S}$: the (finite) set of states;
- $\mathcal{P}$: the state transition probability matrix, where $\mathcal{P}_{ss'} = P(S_{t+1} = s' | S_t = s)$ is the probability of arriving s' from s;
- R: the random variable or function of rewards, and we use R_t to represent the reward at time t;
- γ : the discount factor.

An MRP chain is a Markov chain with values, and we can define *return* as follows.

Return: *Suppose the reward at time* t *is* R_t*, then the return* G_t *is the total discounted reward from time-step* t*:*

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where $\gamma \in (0, 1)$ *is the discount factor.*

There are many reasons to consider discounted rewards. One reason is that uncertainty about the future may not be fully represented. And for the case that the reward is financial, immediate rewards may earn more interest than delayed rewards. Besides, animal and human behavior shows preference for immediate reward.

Value Function: *The state value function* $v(s)$ *of an MRP is the expected return starting from state* s

$$v(s) = \mathbb{E}[G_t | S_t = s].$$

Bellman Equation for MRPs: *The value function* $v(s_t)$ *can be decomposed into two parts: the immediate reward* R_{t+1} *and the discounted value of successor state, which is*

$$v(s) = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s].$$

Proof:

Notice that S is a random variable of states, and s is an instance of states. According to the Adam's Law with extra conditioning, it follows that

$$\mathbb{E}[\mathbb{E}G_{t+1} | S_{t+1}, S_t] | S_t] = \mathbb{E}[G_{t+1} | S_t]. \tag{1}$$

By Markov property, the term $(G_{t+1}|S_{t+1}, S_t)$ equals $(G_{t+1}|S_{t+1})$. Therefore we arrive at

$$\begin{aligned}\mathbb{E}[\mathbb{E}[G_{t+1}|S_{t+1}, S_t]|S_t] &= \mathbb{E}[\mathbb{E}[G_{t+1}|S_{t+1}]|S_t] \\ &= \mathbb{E}[v(S_{t+1})|S_t]. \end{aligned} \tag{2}$$

With (1) and (2), we can make a conclusion that

$$\mathbb{E}[G_{t+1}|S_t] = \mathbb{E}[v(S_{t+1})|S_t]. \tag{3}$$

Then we have

$$\begin{aligned} v(s) &= \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1}|S_t = s] + \gamma\mathbb{E}[G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1}|S_t = s] + \gamma\mathbb{E}[v(S_{t+1})|S_t = s] && \text{(by (3))} \\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \end{aligned}$$

□

We can also derive another form of Bellman equation by the law of total expectation:

$$\begin{aligned} v(s) &= \mathbb{E}[R_{t+1}|S_t = s] + \gamma\mathbb{E}[v(S_{t+1})|S_t = s] \\ &= R_s + \gamma \sum_{s' \in S} \mathbb{E}[v(S_{t+1})|S_t = s, S_{t+1} = s']\mathcal{P}_{ss'} && \text{(by LOTE)} \\ &= R_s + \gamma \sum_{s' \in S} v(s')\mathcal{P}_{ss'}, \end{aligned}$$

Though Bellman equation is elegant, to solve it directly is only possible for small MRPs.

5.2 Markov Decision Process

Markov Decision Process (MDP): *A Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma \rangle$, where*

- $\mathcal{S}$: the (finite) set of states;
- $\mathcal{A}$: the (finite) set of actions;
- $\mathcal{P}$: the state transition probability matrix, where $\mathcal{P}^a_{ss'} = P(S_{t+1} = s'|S_t = s, A_t = a)$ is the probability of arriving s' by taking action a from s;
- R: the random variable or function of rewards, besides the notation we defined in MRP, we also use R^a_s to denote the reward we get by taking action a at state s;
- γ : the discount factor.

An MDP chain is an MRP chain with decisions (actions). It is an environment in which all states are Markovian.

Policy: *A policy π is a distribution over actions given states,*

$$\pi(a|s) = P(A_t = a|S_t = s).$$

A policy can fully define the behavior of an agent, and it depends only on the current state. Similar to MRP, we have the following definitions.

State Value Function: *The state value function $v^\pi(s)$ of an MDP is the expected return starting from state s, and following policy π*

$$v^\pi(s) = \mathbb{E}_\pi[G_t|S_t = s].$$

State-action Value Function: *The state-action value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π, which is*

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a].$$

According to the definition, we have

$$v^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) q^\pi(s, a),$$

$$q^\pi(s, a) = R_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v^\pi(s').$$

Then the relation between $v^\pi(s)$ and $q^\pi(s, a)$ follows that

$$v^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'\in S} \mathcal{P}_{ss'}^a v^\pi\left(s'\right) \right),$$

$$q^\pi(s, a) = R_s^a + \gamma \sum_{s'\in S} \mathcal{P}_{ss'}^a \sum_{a'\in\mathcal{A}} \pi\left(a'|s'\right) q^\pi\left(s', a'\right).$$

And we can rewrite the above equations to get *Bellman expectation equation*.

Bellman Expectation Equation: *The value function can be decomposed into immediate reward plus discounted value of the successor state, that is to say,*

$$v^\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v^\pi(S_{t+1})|S_t = s],$$
$$q^\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q^\pi(S_{t+1}, A_{t+1})|S_t = t, A_t = a].$$

Optimal State Value Function: *The optimal state value function $v_*(s)$ is the maximum value function over all polices:*

$$v_*(s) = \max_\pi v_\pi(s).$$

Optimal State-action Value Function: *The optimal state-action value function $q_*(s, a)$ is the maximum value function over all polices:*

$$q_*(s, a) = \max_\pi q_\pi(s, a).$$

Optimal Policy: *For any Markov Decision Process, there exists an optimal policy π_* that is better than or equal to all other policies. Further, all optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$ and the optimal state-action value function, $q_{\pi_*}(s,a) = q_*(s,a)$.*

We say that an MDP is *solved* when we know the optimal value function. Since the optimal policy can be derived from the optimal value function.

Prediction: *Given an MDP and a policy π, find the value function v^π and q^π.*

Control: *Given an MDP, find the optimal policy π^*.*

Actually, the term *prediction* can be replaced with *evaluation* in many cases. Hope it would not make you confused. For an unknown MDP, we have no idea about its transition matrix and its state, action spaces, thus the methods we mentioned in previous sections fail to work. What we can do is interacting with the environment and collecting episodes.

5.3 Dynamic Programming

Dynamic programming is a method for solving complex problems by breaking them down into subproblems. Specifically, the problems should be characterized by the two properties:

- Optimal substructure, which allows that optimal solution can be decomposed into subproblems;
- Overlapping subproblems, which entails that the solutions for subproblems can be cached and reused.

Obviously, MDPs satisfy both properties. Evaluation and control in MDP can be solved by dynamic programming.

5.3.1 Policy Evaluation

Bellman expectation backup: *Given a policy π, at each iteration $k+1$, for all states $s \in \mathcal{S}$, update $v_{k+1}(s)$ from $v_k(s')$ by*

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right).$$

Clearly, $v_k = v_\pi$ is a fixed point for this update rule because the Bellman equation for v_π assures us of equality in this case. Indeed, the sequence $\{v_k\}$ can be shown in general to converge to v_π as $k \to \infty$ under the same conditions that guarantee the existence of v_π. This algorithm is called iterative policy evaluation.

Algorithm 7 Iterative Policy Evaluation

initialize $v(s)$ arbitrarily for $s \in \mathcal{S}$ except that $v(terminal) = 0$;
input the policy π to be evaluated; the threshold $\theta > 0$ determining the accuracy of the evaluation;
for true **do**:
    $\Delta \leftarrow 0$;
    for each $s \in \mathcal{S}$ **do**:
        $old \leftarrow v(s)$;
        $v(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{ss'}[R^a_s + \gamma v(s')]$;
        $\Delta \leftarrow \max(\Delta, |old - v(s)|)$;
    end for
    if $\Delta < \theta$ **then**:
        break;
    end if
end for

Formally, iterative policy evaluation converges only in the limit, but in practice it is better to be halted short of this.

5.3.2 Policy Iteration

For a deterministic policy $a = \pi(s)$, we can iteratively improve it through the two steps:

- Evaluate the policy π so that get the value function;
- Improve the policy by acting greedily for all $s \in \mathcal{S}$ with respect to the value function:

$$\pi'(s) = \arg\max_a q_\pi(s, a).$$

Proof:

Now we show that such improvement does give us a better policy.

$$\begin{aligned}
v_\pi(s) &\leq \max_a q_\pi(s, a) \\
&= q_{\pi'}(s, \pi'(s)) \\
&= E_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = t] \\
&\leq E_{\pi'}[R_{t+1} + \gamma q_{\pi'}(S_{t+1}, \pi'(S_{t+1}))|S_t = t] \\
&\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_{\pi'}(S_{t+2}, \pi'(S_{t+2}))|S_t = s] \qquad (1) \\
&\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + ...|S_t = s] \\
&= v_{\pi'}(s).
\end{aligned}$$

The step (1) can be derived by Adam's law with extra conditioning and Markov theory. □

When the improvement converges, we have

$$q^{\pi}(s, \pi'(s)) = \max_{a} q^{\pi}(s, a) = q^{\pi}(s, \pi(s)) = v^{\pi}(s),$$

which gives us the optimal policy.

Algorithm 8 Policy Iteration

initialize a policy $\pi(s) \in \mathcal{A}$ arbitrarily for all $s \in \mathcal{S}$;
for true **do**:
    $v \leftarrow$Policy Evaluation for π;
    $stable \leftarrow$ true;
    for each $s \in \mathcal{S}$ **do**:
        $old \leftarrow \pi(s)$;
        $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[R^a_s + \gamma v(s')]$;
        if $old \neq \pi(s)$ **then**:
            $stable \leftarrow$false;
        end if
    end for
    if $stable$ **then**:
        Return π
    end if
end for

It should be noticed that here we are talking about deterministic policy. For stochastic policy, expectation should be introduced.

Generalized Policy Iteration(GPI): *The combination of policy evaluation and policy improvement is called generalized policy iteration.*

In GPI one maintains both an approximate policy and an approximate value function. The value function is repeatedly altered to more closely approximate the value function for the current policy, and the policy is repeatedly improved with respect to the current value function. These two kinds of changes work against each other to some extent, as each creates a moving target for the other, but together they cause both policy and value function to approach optimality.

5.3.3 Value Iteration

One drawback to policy iteration is that each of its iterations involves policy evaluation, which may itself be a protracted iterative computation requiring multiple sweeps through the state set. After the value function converges, the change of the policy will lead to a new round evaluation.

In fact, the policy evaluation step of policy iteration can be truncated in several ways without losing the convergence guarantees of policy iteration. One important special case is when policy evaluation is stopped after just one sweep (one update of each state). This algorithm is called *value iteration*. It can be written as a particularly simple update operation that combines the policy improvement and truncated policy evaluation steps.

Algorithm 9 Value Iteration

initialize $v(s)$ arbitrarily for $s \in \mathcal{S}$ except that $v(terminal) = 0$;
input the threshold $\theta > 0$ determining the accuracy of the evaluation;
for true **do**:
    $\Delta \leftarrow 0$;
    for each $s \in \mathcal{S}$ **do**:
        $old \leftarrow v(s)$;
        $v(s) \leftarrow \max_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{ss'}[R^a_s + \gamma v(s')]$;
        $\Delta \leftarrow \max(\Delta, |old - v(s)|)$;
    end for
    if $\Delta < \theta$ **then**:
        break;
    end if
end for
return $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[R^a_s + \gamma v(s')]$;

We can tell that value iteration is different from policy evaluation in how we update $v(s)$. The update rule in value iteration follows *Bellman optimality equation*.

Bellman Optimality Equation: *The optimal value functions are reached by the Bellman optimality equations*:

$$v^*(s) = \max_a R^a_s + \gamma \sum_{s' \in S} P\left(s'|s,a\right) v^*\left(s'\right).$$

The derivation of the equation is simple. According to the optimal policy, we know

$$v^*(s) = \max_a q^*(s,a). \tag{1}$$

Then from the definition of $q(s,a)$,

$$q^*(s,a) = R^a_s + \gamma \sum_{s' \in S} P\left(s'|s,a\right) v^*\left(s'\right). \tag{2}$$

Therefore we can get the Bellman optimality equation by plugging (2) into (1).

5.3.4 Comparison

The comparison of the two methods are shown as follows.

Policy Iteration: policy evaluation + policy improvement

- starts from an arbitrarily policy and then estimates the *state value* given the policy;
- generates a new policy given the estimated state value;
- converges faster in terms of the number of iterations since it doing a lot more work in each iteration.

Value Iteration: optimal value function + one policy extraction

- updates the value *greedily* (does not care the policy) at each iteration and then, *after* finding the optimal value function, determines a new policy given the optimal value function;
- convergences fast per iteration, but may be far from the true value function.

The summary for evaluation and control in MDP by dynamic programming is shown in Table 1.

Problem	Bellman Equation		Algorithm
Prediction	Expectation	$v^\pi(s) = E_\pi[R_{t+1} + \gamma v^\pi(S_{t+1})\|S_t = s]$	Policy
		$q^\pi(s,a) = E_\pi[R_{t+1} + \gamma q^\pi(S_{t+1}, A_{t+1})\|S_t = t, A_t = a]$	Evaluation
Control	Expectation	$v^\pi(s) = E_\pi[R_{t+1} + \gamma v^\pi(S_{t+1})\|S_t = s]$	Policy
		$q^\pi(s,a) = E_\pi[R_{t+1} + \gamma q^\pi(S_{t+1}, A_{t+1})\|S_t = t, A_t = a]$	Iteration
Control	Optimality	$v^*(s) = \max_a R_s^a + \gamma \sum_{s' \in \mathcal{S}} P(s'\|s,a) v^*(s')$	Value
		$q^*(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} P(s'\|s,a) \max_{a'} q^*(s',a')$	Iteration

Table 1: Dynamic Programming algorithms for MDPs