RL Ch4 Dynammic Programming

Le-Rong Hsu

June 2023

We assume that the environment is a finite MDP, and that the state, action and reward sets, $\mathcal{S}, \mathcal{A}, \mathcal{R}$ are finite, and that its dynamics are given by a set of probabilities $p(s', r \mid s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}, r \in \mathcal{R}, s' \in \mathcal{S}^+$ ($\mathcal{S}^+ = S+$ a terminal state if the problem is episodic).

1 Policy Evaluation (Prediction)

Recall from chapter 3 that, for all $s \in \mathcal{S}$,

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_t + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_t + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s')\right], \end{aligned}$$

where $\pi(a \mid s)$ is the probability of taking action a in state s under policy π. The existence and uniqueness of v_π is guaranteed as long as $\gamma < 1$ or eventual termination is guaranteed from all states.

If the dynamics is completely known, then the above is a $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns ($v_\pi(s)$ for all $s \in \mathcal{S}$).

Consider a sequence of approximation value functions $v_0, v_1, ...,$, each mapping $\mathcal{S}^+$ to $\mathbb{R}$. v_0 is chosen arbitrarily and for each k,

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_k(s_{t+1}) \mid S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s')\right] \end{aligned}$$

In general, $\{v_k\}$ can be shown to converge to v_π as $k \longrightarrow \infty$. This algorithm is called *iterative policy evaluation*.

Algorithm 1 Iterative Policy Evaluation for estimating $V \approx v_\pi$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation.
Initialize $V(s)$ for all $s \in \mathcal{S}^+$, arbitrarily except that $V = 0$
$\Delta \leftarrow 0$
while $\Delta >= \theta$ **do**
  for each $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
end while

2 Policy Improvement

One reason for computing the value function for a policy v_π is to help find better policies.

Given π and v_π. For some state s, we would like to know whether or not we should change the policy to choose an action $a \neq \pi(s)$. $v_\pi(s)$ tells us how good it is, but how do we know it would be better or worse to change to the new policy? One way to answer this question is to consider selecting an action a and thereafter follow an existing policy π:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}[R_{t+1} + \gamma v_\pi(s_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s')\right] \end{aligned} \tag{1}$$

It's natural to ask whether $q_\pi(s, a)$ is greater than or less than $v_\pi(s)$. If it is better to select a once in state s and thereafter follow π than it would be follow π all the time, then one would expect it to be better to select a every time s is encountered.

That the above is true is a special case of the *policy improvement theorem*. Let π and π' be any pair of deterministic policies. We say that π' is better than π if

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \tag{2}$$

i.e.

$$v_{\pi'}(s) \geq v_\pi(s) \tag{3}$$

Remark. If for the two policies π and π' and for some state s, we have $\pi'(s) = a \neq \pi(s)$ and $q_{\pi'}(s, a) > v_\pi(s)$ with $q_{\pi'}(s', a) = v_\pi(s')$ for all $s' \neq s$, then we still say that π' is better than π.

The idea behind the proof of policy improvement is easy to understand. We keep expand the q_π side with and reapplying (2) until we get $v_{\pi'}$.

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma v_\pi(S_{t+2}) \mid S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) \mid S_{t+1}] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s] && \text{(iterated expectation)} \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi S_{t+3}) \mid S_t = s] \\
&\vdots \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s] \\
&= v_{\pi'}(s)
\end{aligned}
$$

This is the policy improvement at a single state. It's a natural extension to consider changes at all states where we select at each state the best action according to $q_\pi(s, a)$. In other words, consider the *greedy* policy π' given by

$$
\begin{aligned}
\pi'(s) &= \arg\max_a q_\pi \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] && (*) \\
&= \arg\max_a \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s')\right]
\end{aligned}
$$

The greedy policy takes the action that looks best in the short term. By construction, the greedy policy meets the condition (2), so it's as good as or better than the original policy.

Suppose that the new greedy policy π' is as good as, but not better than the old policy π. Then $v_\pi = v_{\pi'}$ and from (*), we have

$$
\begin{aligned}
v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_{\pi'}(s')\right]
\end{aligned}
$$

But this is actually the Bellman equation, and therefore, $v_{\pi'}$ must be v_*, and both π and π' are optimal policies. Thus, policy improvement must give us a strictly better policy except when the original policy is already optimal.

3 Policy Iteration

Once a policy, π, has been improved using v_π to yield a better policy π', we can compute $v_{\pi'}$ and improve π' again to yield a better π''.

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \dots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_{\pi_*}$$

Because a finite MDP has only a finite number of policies, this process must converge to an optimal policy and the optimal value function in a finite number of iteration.

Algorithm 2 Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization: $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$
2. Policy Evaluation:

$\Delta \leftarrow 0$
while $\Delta >= \theta$ **do**
  for each $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
end while

3. Policy Improvement:

policy-stable $\leftarrow$ *true*
for each $s \in \mathcal{S}$ **do**
  old-action $\leftarrow \pi(s)$
  $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$
  If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
end for
If the policy is stable, then return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2.

4 Value Iteration

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 sweep: one update of each state). It can be written as a particularly simple update operation that combines the policy improvement and truncated policy evaluation steps:

$$v_{k+1}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \tag{4}$$

$$= \max_a \sum_{s',r} p(s', r \mid s, r)[r + \gamma v_k(s')] \tag{5}$$

for all $s \in \mathcal{S}$. For arbitrary v_0, the sequence $\{v_k\}$ can be shown to converge to v_* provided the existence of v_*. Note that value iteration is obtained simply by turning the Bellman optimality equation into an update rule. Also note how the value iteration update is identical to the policy evaluation update except that it requires the maximum to be taken over all actions.

Algorithm 3 Value Iteration for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$
$\Delta \leftarrow 0$
while $\Delta >= \theta$ **do**
  for each $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s)]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
end while
Output a deterministic policy, $\pi \approx \pi_*$, such that

$$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$$

5 Asynchronous Dynamic Programming

If the state set is very large, then even a single sweep (one update of each state) can be prohibitively expensive. Hence, it is natural to ask whether we can select some states that we updates more frequently and the others updated less frequently. The answer is true and these algorithms are called asynchronous DP algorithms. *Asynchronous* DP algorithms are in-place DP algorithms that are not organized in terms of systematic sweeps of the state set. These algorithms update the values of states in any order whatsover, **using whatever values of other states happen to be available**.

However, to converge correctly, an asynchronous DP algorithm must continue to update the values of all the states: it can't ignore any state "after some point in the computation". Asynchronous DP algorithm allow great flexibility in selecting states to update.

Avoiding sweeps doe not necessarily mean that we can get away with less computations. It just means that an algorithm does not need to get locked into any hopelessly long sweep before it can make progress improving a policy. We can try to take advantage of this flexibility by selecting the states to which we apply updates so as to improve the algorithm's rate of progress.

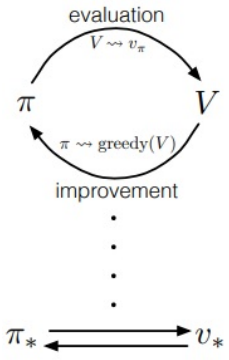
Remark. In-place means that the algorithm does not use extra space for manipulating the input but may require a small though non-constant extra space for its operation.

(the textbook does not provide any example of asynchronous DP algorithm)

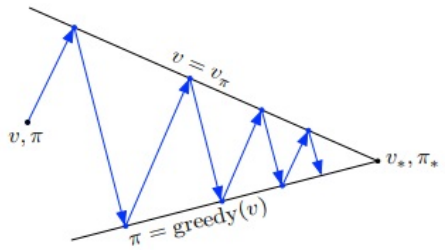
6 General Policy Iteration

Policy iteration consists of two simultaneous, interacting process, one making the value function consistent with the current policy (policy evaluation), and the other making the policy greedy with respect to the current value function (policy improvement). These two processes alternate, each completing before the other begins, but this is not necessary. As long as both processes continue to update all states, the ultimate result is typically the same: convergence to the optimal value function and optimal policy.

We use the term *generalized policy iteration* (GPI) to refer to the general idea of letting policy-evaluation and policy-improvement. Almost all reinforcement learning methods are well described as GPI. That is, all have identifiable policies and value functions with the policy always being improved with respect to the value function and the value function always being driven toward the value function for the policy. If both the evaluation process and the improvement process stabilize, then the value function and policy must be optimal.



One may also think of the interaction between the evaluation and improvement processes in GPI. The arrows in the diagram correspond to the behavior of policy iteration in that each one could also take smaller, incomplete steps toward each goal. The two processes together achieve the overall goal of optimality even though neither is attempting to achieve it directly.



7 Efficiency of Dynamic Programming

DP may not be practical for very large problems, but compared with other methods solving MDPs, DP methods are actually quite efficient. If n and k denote the number of states and actions, this means that a DP method takes a number of computational operations that is less than some polynomial function of n and k. A DP method is guaranteed to find an optimal policy in polynomial time even though the total number of policies is k^n.

Linear programming methods can also be used to solve MDPs, but linear programming methods become impractical at a much smaller number of states than do DP methods. For the largest problems, only DP methods are feasible.

In practice, DP methods can be used with today's computers to solve MDPs with millions of states. Both policy iteration and value iteration are widely used, and it is not clear which is better. In practice, these methods usually converge much faster than their theoretical worst-case run times, particularly if they are started with good initial value functions or policies.

On problems with large state spaces, *asynchronous* DP methods are often preferred. A synchronous method requires computation and memory for every state to complete one sweep. This is sometimes impractical, yet the problem is still solvable because relatively few states occur along optimal solution trajectories. Asynchronous methods and other variations of GPI can be applied in such cases.

8 Summary

This chapter introduces basic algorithms for solving finite MDPs. *Policy evaluation* refers to iterative computation of the value function for a given policy. *Policy improvement* refers to the computation of an improved policy given the value function for that policy. Putting these two computations together, we obtain *policy iteration* and *value iteration*, the most popular DP methods.

Classical DP methods operate in sweeps through the state set, performing an *expected update* operation on each state. Each operation updates the (expected) value of one state based on the (expected) values of all possible successor states and their probabilities of occurring. When the updates no longer result in any changes in value, convergence has occurred to values that satisfy the corresponding Bellman equation.

For GPI, one process takes the policy as given and performs some form of policy evaluation, changing the value function to be more like the true value function for the policy. The other process takes the value function as given and performs some form of policy improvement, changing the policy to make it better, assuming that the value function is its value function.