

4 Dynamic Programming

The term Dynamic Programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given perfect model of the environment as a Markov Decision Process (MDP). DP methods tend to be computationally expensive and we often don't have a perfect model of the environment, so they aren't used in practice. However, they provide useful theoretical basis for the rest of reinforcement learning.

Unless stated otherwise, will assume that the environment is a finite MDP. If the state or action space is continuous, then we will generally discretise it and apply finite MDP methods to the approximated problem.

The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies. We use DP and the Bellman equations to find optimal value functions.

4.1 Policy Evaluation (Prediction)

We can use the Bellman equation for the state-value function v_π to construct an iterative updating procedure.

Iterative Policy Evaluation

Consider a sequence of approximate value functions v_0, v_1, v_2, \dots each mapping \mathcal{S}^+ to \mathbb{R} . The initial approximation, v_0 , is chosen arbitrarily (except that the terminal state, if any, must be given value 0), and each successive approximation is obtained by using the Bellman equation for v_π as an update rule:

$$v_{k+1} \doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \quad (26)$$

$$= \sum_a \pi(s|a) \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')] \quad (27)$$

Clearly, $v_k = v_\pi$ is a fixed point. The sequence $\{v_k\}$ can be shown in general to converge to v_π as $k \rightarrow \infty$ under the same conditions that guarantee the existence of v_π . This algorithm is called *iterative policy evaluation*. This update rule is an instance of an *expected update* because it performs the updates by taking an expectation over all possible next states rather than by taking a sample next state.

4.2 Policy Improvement

Policy Improvement Theorem

Let π, π' be any pair of deterministic policies, such that

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \quad \forall s \in \mathcal{S}. \quad (28)$$

That is, π' is at least as good as π . Then we have (shown below)

$$v_{\pi'}(s) \geq v_\pi(s) \quad \forall s \in \mathcal{S} \quad (29)$$

so π' gives at least as good (expected) return as π .

The argument below also shows that if $q_\pi(s, \pi'(s)) > v_\pi(s)$ at any s , then there is at least one s for which $v_{\pi'}(s) > v_\pi(s)$.

proof:

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\
&= v_{\pi'}(s)
\end{aligned}$$

Policy Improvement Algorithm

Now consider a policy that is greedy with respect to $q_\pi(s, a)$. Define

$$\pi'(s) = \operatorname{argmax}_a q_\pi(s, a) \quad (30)$$

$$= \operatorname{argmax}_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \quad (31)$$

$$= \operatorname{argmax}_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')]. \quad (32)$$

Now we can use v_π to get $\pi' \geq \pi$, then use $v_{\pi'}$ to get *another* policy. (In the above, ties are broken arbitrarily when the policy is deterministic. If the policy is stochastic, we accept any policy that assigns zero probability to sub-optimal actions.)

Note that by construction

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

therefore

$$v_{\pi'} \geq v_\pi$$

so we get from this process a monotonically increasing sequence of policies.

Note also that if π' is as good as π then $v_{\pi'} = v_\pi$ and $\forall s \in \mathcal{S}$

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

which is the Bellman optimality condition for v_* , so both π and π' are optimal. This means that policy improvement gives a strictly better policy unless the policy is already optimal.

The policy improvement theorem holds for stochastic policies too, but we don't go into that here.

4.3 Policy Iteration

We can exploit policy improvement iteratively to get the policy iteration algorithm.

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
Loop:
 $\Delta \leftarrow 0$
 Loop for each $s \in \mathcal{S}$:
 $v \leftarrow V(s)$
 $V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s)) [r + \gamma V(s')]$
 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
 until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)
3. Policy Improvement
 policy-stable \leftarrow *true*
 For each $s \in \mathcal{S}$:
 old-action $\leftarrow \pi(s)$
 $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$
 If *old-action* $\neq \pi(s)$, then *policy-stable* \leftarrow *false*
 If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

A finite MDP has only a finite number of policies (as long as they are deterministic, of course) so this process is guaranteed to converge.

4.4 Value Iteration

Policy iteration can be slow because each iteration involves running the entire policy evaluation until convergence.

It turns out that one can truncate the policy evaluation step of policy iteration in many ways without losing convergence guarantees. One special case of this is *value iteration*, where we truncate policy evaluation after only one update of each state. This algorithm converges to v_* under the same conditions that guarantee the existence of v_* .

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 $V(\text{terminal}) = 0$

Loop:
 $\Delta \leftarrow 0$
 Loop for each $s \in \mathcal{S}$:
 $v \leftarrow V(s)$
 $V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$
 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
 until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
 $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$

Note the \max_a in the assignment of $V(s)$, since we only one sweep of the state space and then choose the greedy policy.

It may be more efficient to interpose multiple policy evaluation steps in between policy improvement iterations, all of these algorithms converge to an optimal policy for discounted finite MDPs.

4.5 Asynchronous Dynamic Programming

The DP methods that we have described so far all involve a full sweep of the state space on each iteration. This is potentially a very costly procedure.

Asynchronous DP algorithms update the values in-place and cover states in any order whatsoever. The values of some states may be updated several times before the values of others are updated once. To converge correctly, however, an asynchronous algorithm must continue to update the values of all the states: it can't ignore any state after some point in the computation.

Asynchronous DPs give a great increase in flexibility, meaning that we can choose the updates we want to make (even stochastically) based on the interaction of the agent with the environment. This procedure might not reduce computation time in total if the algorithm is run to convergence, but it could allow for a better rate of progress for the agent.

4.6 Generalised Policy Iteration

We use the term *generalised policy iteration* (GPI) to refer to the general idea of letting policy evaluation and policy improvement processes interact, independent of the granularity and other details of the two processes. Almost all reinforcement learning methods are well described as GPI, including the policy iteration algorithms we have discussed in this section. GPI works via the competing but complementary nature of the two processes. In some cases it can be guaranteed to converge.

4.7 Efficiency of Dynamic Programming

If we ignore a few technical details, then the (worst case) time DP methods take to find an optimal policy is polynomial in the number of states and actions. Compare this to the searching the states directly, which is exponential.