Dynamic Programming

Sunmook Choi felixchoi@korea.ac.kr

December 14, 2023

Dynamic programming is a general approach to solving problems that have a recursive structure by breaking them down into simpler subproblems and reusing the solutions to these subproblems to solve the original problem. In reinforcement learning, dynamic programming algorithms are used to compute optimal policies and optimal value functions. There are two main types of dynamic programming algorithms: value iteration and policy iteration. Implementation of these algorithms for some MDP models can be found in my GitHub repository: https://github.com/smfelixchoi/MATH-DRL-study/tree/main/1.Dynamic_Programming

1 Value Iteration

Value iteration is a dynamic programming algorithm that finds an optimal policy by estimating an optimal state-value function directly using the Bellman optimality equation. Recall that the Bellman optimality equation is the recursive Bellman equation for optimal value function:

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma\, v_*(s')] \tag{1.1}$$

Algorithm 1 (Value Iteration)**.** *Value iteration algorithm is a method to estimate the optimal state-value function v_* iteratively by following the steps below:*

1. *Initialize $V_0(s)$, for all $s \in \mathcal{S}$, arbitrarily except for $V(terminal) = 0$.*
2. *Update $V_{k+1}(s)$ iteratively from all $V_k(s')$ implementing the following rule until convergence:*
   a. *synchronous backups: compute $V_{k+1}(s)$ for all s and update it simultaneously.*
   b. *asynchronous backups: compute $V_{k+1}(s)$ for one s and update it immediately.*

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')] \tag{1.2}$$

3. *Compute the optimal policy $\pi_* = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V^*(s)\big]$.*

In the algorithm, the convergence at kth step implies that, for sufficiently small threshold ϵ,

$$|V_{k+1}(s) - V_k(s)| < \epsilon \quad \text{for all } \ s \in \mathcal{S}. \tag{1.3}$$

We can prove that the algorithm ensures the convergence, that is, $V_k \to v_*$ as $k \to \infty$. The proof will be written in the end of this chapter. For practical reasons, we also state a pseudocode of the algorithm in below.

Algorithm 1.1 Value Iteration

Parameter: a small threshold $\theta > 0$ determining accuracy of estimation.

Initialize $V(s)$, for all $s \in \mathcal{S}$, arbitrarily except that $V(terminal) = 0$.

while True **do**
  $\Delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
  if $\Delta < \theta$ **then**
    break
  end if
end while
return a deterministic policy, $\pi \approx \pi_*$, such that

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

However, there are two disadvantages of the algorithm about its convergence.

1. The action argument inducing the maximum at each state rarely changes, so the policy often converges long before the values converge.
2. It is slow because its complexity is $O(S^2A)$ per each iteration and it needs a lot of iterations to reach convergence.

2 Policy Iteration

Another algorithm of dynamic programming to find an optimal policy is called *policy iteration*. Policy iteration have two iterative steps: *policy evaluation* and *policy improvement*. Policy evaluation step computes the value function for a given policy, and policy improvement step computes a better policy based on the current value function. These two steps are iterated until convergence to obtain an optimal policy and the optimal value function. The main difference from value iteration is that, in the policy evaluation step, a policy is fixed and the update rule is based on Bellman expectation equation.

Algorithm 2 (Policy Iteration)**.** *Policy iteration algorithm is a method to find an optimal policy by taking two steps, policy iteration and policy improvement, iteratively until convergence.*

1. **Policy Evaluation**: *Computing V^π from a fixed deterministic policy π.*
   (a) *Initialize $V_0(s)$, for all $s \in \mathcal{S}$, arbitrarily except for $V(\text{terminal}) = 0$.*
   (b) *Update every $V_{k+1}(s)$ from all $V_k(s')$ until convergence to $V^\pi(s)$.*

$$V_{k+1}(s) = \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V_k(s')] \tag{2.1}$$

2. **Policy Improvement**: *Improving π to π' by greedy policy based on V^π.*

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V^\pi(s')] = \arg\max_a Q^\pi(s, a) \tag{2.2}$$

3. *If $\pi' \neq \pi$, then go back to Policy Evaluation phase. Otherwise, π' is an optimal policy.*

We can prove that the policy evaluation step ensures the convergence, that is, $V_k \to v_\pi$ as $k \to \infty$ for a given policy π. The proof will be also given in the end of this chapter. Furthermore, the policy improvement step indeed improves the policy: in this step, for the updated policy $\pi'(s) = \arg\max_a Q^\pi(s, a)$ for all $s \in \mathcal{S}$, we have

$$Q^\pi(s, \pi'(s)) \geq \sum_{a \in \mathcal{A}} \pi(a|s)\, Q^\pi(s, a) = V^\pi(s), \tag{2.3}$$

since the maximum is always greater than the average. Then, the following theorem, *Policy Improvement Theorem*, ensures that $\pi' \geq \pi$. In the theorem, the statement is in more general situation where π' is stochastic.

Theorem 3 (Policy Improvement Theorem)**.** *Let π and π' be two policies and define*

$$q_\pi(s, \pi') = \mathbb{E}_{a \sim \pi'(\cdot|s)}[q_\pi(s, a)],$$

If $q_\pi(s, \pi') \geq v_\pi(s)$ for all $s \in \mathcal{S}$, then we have

$$v_{\pi'}(s) \geq v_\pi(s), \quad \forall s \in \mathcal{S}, \tag{2.4}$$

that is, $\pi' \geq \pi$.

Proof. Recall the following equation:

$$q_\pi(s, a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma\, v_\pi(s')] = \mathbb{E}_\pi[R_{t+1} + \gamma\, v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$

Then we have

$$\begin{aligned} q_\pi(s, \pi') &= \sum_{a \in \mathcal{A}} \pi'(a|s)\, q_\pi(s, a) \\ &= \sum_{a \in \mathcal{A}} \pi'(a|s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma\, v_\pi(s')] \\ &= \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma\, v_\pi(S_{t+1}) \mid S_t = s\big] \end{aligned}$$

Here, the expectation is no longer conditioned on the action because we marginalized over the action space. Then we obtain the following:

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

This completes the proof. □

The algorithm obtains a better policy by repeating the policy evaluation step and the policy improvement step alternatively. For finite MDP case, the number of possible policies is finite, so that the algorithm will give you an optimal policy in finite iterations. Let's see why the output policy of the algorithm is optimal.

In the policy improvement step, assume that the update policy π' is equal to the old policy π, that is,

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

where V^π is obtained from the policy evaluation satisfying

$$V^\pi(s) = \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma\, V^\pi(s')\big]. \tag{2.5}$$

However, since the policy π' (which is equal to π) is the greedy deterministic policy, the equation (2.5) can be restated as

$$V^\pi(s) = \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma\, V^\pi(s')\big]. \tag{2.6}$$

Actually, the equation (2.6) is exactly the Bellman optimality equation. Therefore, since the value function of the policy π is optimal, we can conclude that the policy π is an optimal policy.

A pseudocode of the algorithm is showed below for practical reasons. Compared to value iteration algorithm, the complexity of policy evaluation step is $O(S^2)$ per each iteration. Moreover, we do not have to continue the algorithm redundantly since the algorithm directly finds an optimal policy instead of the optimal value function.

Algorithm 2.1 Policy Iteration

Parameter: a small threshold $\theta > 0$ determining accuracy of estimation.

Initialize $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

Policy Evaluation

while True **do**
    $\Delta \leftarrow 0$
    for $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
    if $\Delta < \theta$ **then**
        break
    end if
end while

Policy Improvement

policy-stable $\leftarrow$ True
for $s \in \mathcal{S}$ **do**
    old-action $\leftarrow \pi(s)$
    $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    if *old-action*$\neq \pi(s)$ **then**
        policy-stable $\leftarrow$ False
    end if
end for
if *policy-stable* **then**
    return $V \approx v_*$ and $\pi \approx \pi_*$
else go to *Policy Evaluation step*
end if

3 Proofs of Algorithms

References

[1] Andrew Barto and Richard S. Sutton, *Reinforcement Learning: An Introduction* (2nd ed.), The MIT Press, 2018.