Dynamic Programming

Sunmook Choi felixchoi@korea.ac.kr

August 6, 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.

1 Value Iteration

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

Algorithm 1 (Value Iteration)**.** *Value iteration algorithm is a method to estimate an optimal state-value function v_* iteratively.*

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.*

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')] \tag{1.1}$$

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

The convergence in the algorithm means that, for sufficiently small threshold ϵ,

$$|V_{k+1}(s) - V_k(s)| < \epsilon \quad \text{for all } \ s \in \mathcal{S}. \tag{1.2}$$

The convergence of the algorithm, i.e., $V_k \to V^*$ as $k \to \infty$, is proved below.

Proof. □

We also state a pseudocode of the algorithm in below.

Value Iteration

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation.
Initialize $V(s)$, for all $s \in \mathcal{S}$, arbitrarily except that $V(terminal) = 0$.

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

Output 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.

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

Policy iteration have two iterative steps: policy evaluation and policy improvement. Policy evaluation computes the value function for a given policy, and policy improvement computes a better policy based on the current value function. These two steps are iterated until convergence to obtain the optimal policy and 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 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 $V(terminal) = 0$.
   (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.

In the policy evaluation step, given a policy π, we can prove that $V_k \to V^\pi$ as $k \to \infty$.

Proof. □

Furthermore, we can see that the policy improvement step indeed improves the policy: in this step, since the updated policy is $\pi'(s) = \arg\max_a Q^\pi(s, a)$, we have

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

Then, the following theorem ensures that $\pi' \geq \pi$.

Theorem 3 (Policy Improvement Theorem)**.** *Let π and π' be two policies. If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all $s \in \mathcal{S}$, then we have*

$$v'_\pi(s) \geq v_\pi(s) \tag{2.4}$$

for all $s \in \mathcal{S}$, that is, π' is better than π.

Proof. □

We now showed that the policy iteration finds a better policy. For finite MDP case, the number of possible policies is finite, so that the algorithm will give you an optimal policy in finite iterations. A pseudo-code of the algorithm is showed below.

Policy Iteration

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

Policy Improvement

policy-stable $\leftarrow$ True

for $s \in \mathcal{S}$:

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* $\neq \pi(s)$:

policy-stable $\leftarrow$ False

if *policy-stable*:

stop and return $V \approx v_*$ and $\pi \approx \pi_*$

else go to *Policy Evaluation* step.

Compared to value iteration algorithm, the complexity of policy evaluation step is $O(S^2)$ per each iteration. Also, we do not have to continue the algorithm redundantly since the change of policies converges faster than the change of value functions.

3 Examples