Reinforcement Learning

Lesson 4: Dynamic Programming

Edoardo Fazzari, 2022

Overview

- Policy Evaluation (Prediction)
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous Dynamic Programming
- Generalized Policy Iteration
- Efficiency of Dynamic Programming

Dynamic Programming

An introduction

- The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP)
- Classical DP algorithms are of limited utility in RL both because of their assumption and because of their great computational expense
- We assume that the environment is a finite MDP

Policy Evaluation (Prediction)

Policy Evaluation

Iterative solution methods

- We already know the Bellman Equation
- If the environment's dynamic are completely known and the Bellman Equation is a system of $|\mathcal{S}|$ simultaneous linear equation $|\mathcal{S}|$ in unknowns
- For our purposes, *iterative solution methods* are the suitable
  - Consider a sequence of approximate value functions $v_0, v_1, v_2, \dots$
  - The initial approximation v_0 is chosen arbitrarily
  - Each successive approximation is obtained by using the Bellman equation as an update rule:

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big] \end{aligned} \qquad (4.5)$$

Iterative Policy Evaluation considerations

- $v_k = v_\pi$ is a fixed point for this update rule because the Bellman equation for v_π assures us of equality
- 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

- To produce each successive approximation, v_{k+1} from v_k, iterative policy evaluation applies the same operation to each state *s*: It replaces
    - the old value of *s* with a new value obtained form the old values of the successor state of *s*
    - And, the expected immediate reward, along all the one-step transitions possible under the policy being evaluated (*Expected Update*)

Iterative Policy Evaluation

Pseudocode (estimating $V \approx v_\pi$)

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(terminal)$ to 0

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

Policy Improvement

Policy Improvement Theorem

Part 1

- Let's consider again the Bellman equation

$$
\begin{aligned}
q_\pi(s,a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma v_\pi(s')\Big].
\end{aligned} \tag{4.6}
$$

- The key criterion is whether this is greater than or less than $v_\pi(s)$. If it is greater:
    - that is, if it is better to select *a* once in *s* and thereafter follow π than it would be to follow π all the time
    - then one would expect it to be better still to select *a* every time *s* is encountered, and that the new policy would in fact be a better one overall

Policy Improvement Theorem

Part 2

- If that is true, we have a general result called *policy improvement theorem*
    - Let π and π' be any pair of deterministic policies such that, for all $s \in \mathcal{S}$

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \qquad (4.7)$$

- Then the policy π' must be as good as, or better than, π.
    - It must obtain greater or equal expected return from all states $s \in \mathcal{S}$

$$v_{\pi'}(s) \geq v_\pi(s) \qquad (4.8)$$

- *NOTE THAT*: If there is a strict inequality of (4.7) at any state, then there must be strict inequality of (4.8) at that state

Greedy Policy

- So far we have seen how, given a policy and its value function, we can easily evaluate a change in the policy at a single state
- It is a natural extension to consider changes at *all* states, selecting at each state the action that appears best according to $q_\pi(s, a)$ (*the greedy policy* π')

$$
\begin{aligned}
\pi'(s) &\doteq \underset{a}{\arg\max}\, q_\pi(s,a) \\
&= \underset{a}{\arg\max}\, \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \underset{a}{\arg\max} \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big]
\end{aligned}
\tag{4.9}
$$

- By construction, it meets the conditions of the policy improvement theorem
    - so we know that it is as good as, or better than, the original policy

Policy Improvement
Definition

- The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called *policy improvement*

Optimal Policy

- Suppose the new greedy policy, π', is as good as, but not better than, the old policy π. Then $v_\pi = v_{\pi'}$
- From (4.9) it follows that for all $s \in \mathcal{S}$:

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

- Which is the same of the Bellman optimality equation. Therefore:
  - v'_π must be v_*
  - π and π' must be optimal policies

Policy Iteration

Policy Iteration

- Based on the previous results, a sequence of monotonically improving policies and value functions can be obtained:

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

Policy Evaluation

Policy Improvement

- Because a finite MDP has only a finite number of deterministic policies, this process must converge to an optimal policy and the optimal value function in a finite number of iterations
- This way of finding an optimal policy is called *policy iteration*

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 $s \in \mathcal{S}$; $V(\textit{terminal}) \doteq 0$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\qquad v \leftarrow V(s)$
   $\qquad V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   $\quad$ 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

Drawback

- Each iterations involves policy evaluation, which may itself be a protracted iterative computation requiring multiple sweeps through the state set
- If policy evaluation is done iteratively
    - Then convergence exactly to v_π occurs only in the limit

- *Must we wait for exact convergence, or can we stop short of that*?

Value Iteration

Value Iteration

Part 1

- The policy evaluation step 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*

Value Iteration

Part 2

- This algorithm can be written as a simple update operation that combines the policy improvement and truncated policy evaluation steps:

$$
\begin{aligned}
v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big]
\end{aligned} \tag{4.10}
$$

- For all $s \in \mathcal{S}$
- For arbitrary v_0, the sequence $\{v_k\}$ can be shown to converge 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 $V(\mathit{terminal}) = 0$

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

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

Value Iteration

Final considerations

- Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement
- Faster convergence is often achieved by interposing multiple policy evaluation sweeps between each policy improvement sweep

Asynchronous Dynamic Programming

Drawback of DP methods

- They involve operations over the entire state set of the MDP
  - They require sweeps of the state set
  - If the state set is very large, then even a single sweep can be prohibitively expensive

Asynchronous DP

- Asynchronous DP algorithms are in-place iterative 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 whatsoever, using values of there states happen to be available
- The value of some states may be updated several times before the values of others are updated once
- To converge correctly, it must continue to update the values of all the states
- Asynchronous DP algorithms allow great flexibility in selecting states to update

Avoiding sweeps

- Avoiding sweeps does not necessarily mean that we can get away with less computation
- It just means that an algorithm does not need to get locked into any hopeless 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
- We can try to order the updates to let value information propagate from state to state in an efficient way
    - Some states may not need their values updated as often as others
    - We might even try to skip updating states entirely if they are not relevant to optimal behavior

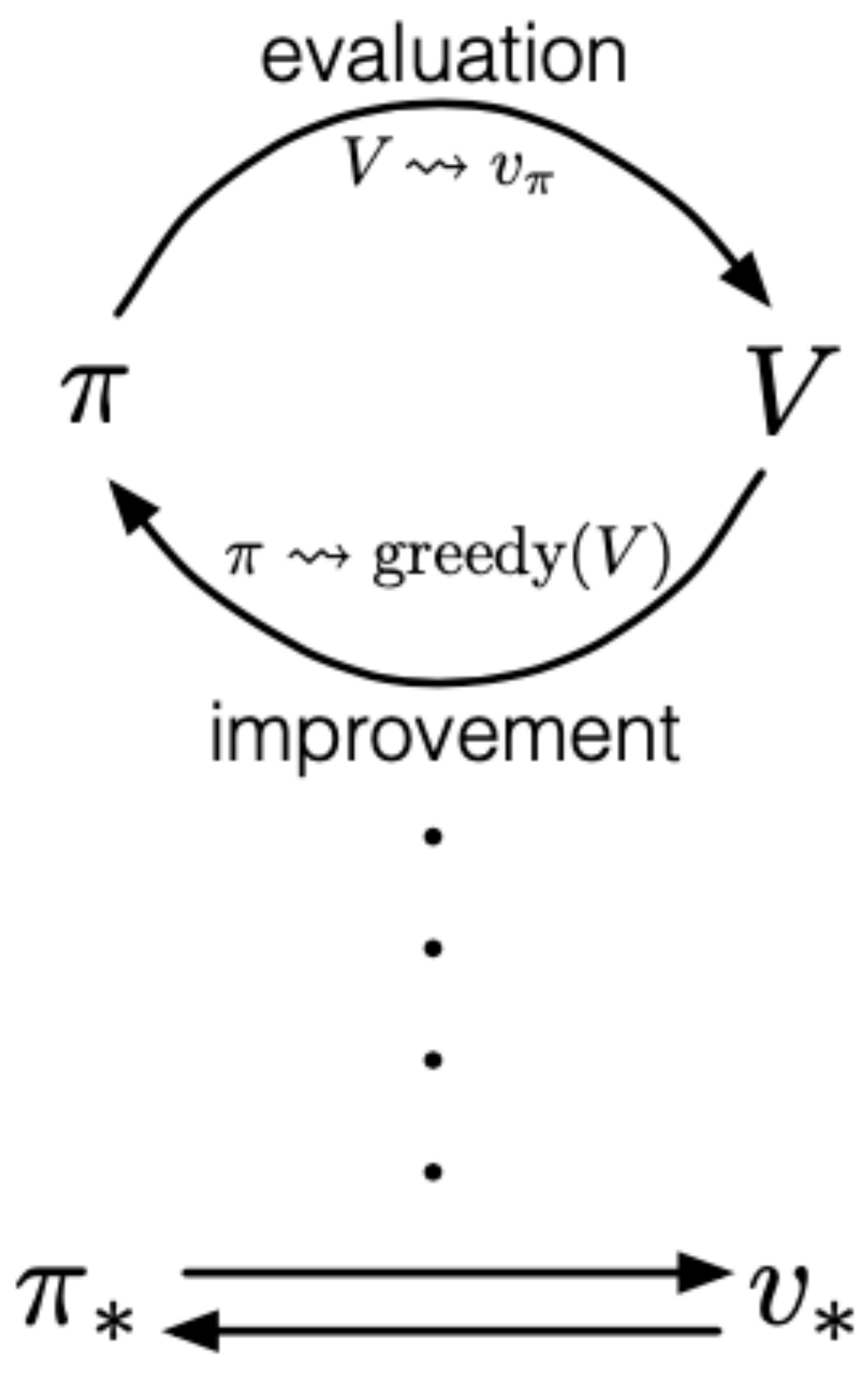
Intermix computation

- Asynchronous algorithms make it easier to intermix computation with real-time interaction
- To solve a given MDP, we can run an iterative DP algorithm *at the same time that an agent is actually experiencing the MDP*
    - The agent's experience can be used to determine the state to which the DP algorithm applies its updates
    - At the same time, the latest value and policy information from the DP algorithm can guide the agent's decision making

Generalized Policy Iteration

Generalized Policy Iteration

- We use the term *generalized policy iteration (GPI)* to refer to the general idea of letting policy-evolution and policy-improvement processes interact, independent of the granularity and other details of the two process
- Almost all RL methods are well described as GPI:
    - 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



Competing and Cooperating

- The evaluation and improvement processes in GPI can be viewed as both competing and cooperating
- They compete in the sense that they pull in opposite directions
    - Making the policy greedy with respect to the value function typically makes the value function incorrect for the changed policy
    - Making the value function consistent with the policy typically causes that the policy no longer to be greedy
- In the long run, however, these two processes interact to find a single joint solution: the *optimal value function* and an *optimal policy*

Efficiency of Dynamic Programming

Some considerations

- DP may not be practical for very large problems, but compared with other methods for solving MDPs, DP methods are actually quite efficient
- If we ignore a few technical details, in the worst case, the time that DP methods take to find an optimal policy is *polynomial* in the number of states and actions
    - If they are started with good initial value functions or policies, they usually converge much faster than their theoretical worst-case run times
    - DP is comparatively better suited to handling large state spaces than competing methods such as *direct search* and *linear programming*
- In practice, DP methods can be used with today's computers to solve MDPs with millions of states
- Both policy interaction and value iteration are widely used, and it is not clear which, if either, is better in general

Problems with large spaces

- On problems with large state spaces, *asynchronous DP methods* are often preferred
    - To complete even one sweep of a synchronous method requires computation and memory for every state
    - For some problems, even this much memory and computation is impractical
        - Yet the problem is still potentially solvable because relatively few states occur along optimal solution trajectories

Bibliography:

Reinforcement Learning An Introduction (Second Edition), R. S. Sutton & A. G. Barto