Lecture 4: Dynamic Programming (DP)

Tianpei Xie

Aug 2nd., 2022

Contents

1 Policy evaluation vs. Control **2**

2 Policy evaluation (Prediction) **3**

3 Policy Iteration (Control) **4**

4 Value iteration **7**

5 Generalized Policy Iteration **8**

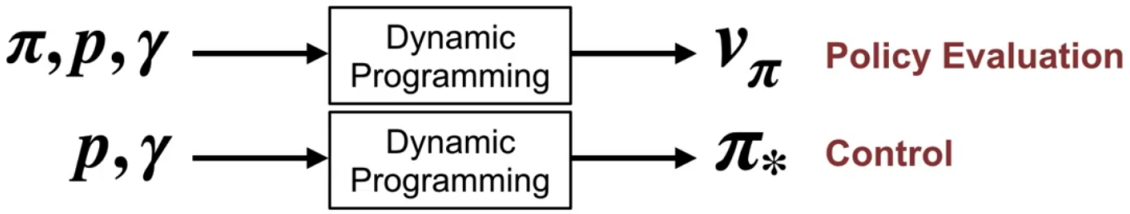

Figure 1: Policy evaluation (Prediction) vs. Control (Policy improvement)

1 Policy evaluation vs. Control

In reinforcement learning, there are two major distinct tasks: *policy evaluation* and *control*:

- **Policy evaluation** or ***Prediction*** is the task of determining the value function for a specific policy. Note that the value function is a prediction of future rewards, in terms of expected returns. Thus estimation of value function is a task of predicting future rewards. The prediction task is an *unsupervised learning task*, since no human label is required and we just need to wait for the rewards.
- **Policy Improvement** or ***Control*** is the task of **finding a policy** to obtain as much reward as possible. In other words, finding a policy which *maximizes* the value function.

Control is the ultimate goal of reinforcement learning. But the task of policy evaluation is usually a necessary first step. It's hard to improve our policy if we don't have a way to assess how good it is. This chapter, we will look at a collection of algorithms called ***dynamic programming (DP)*** for solving both policy evaluation and control problems.

Dynamic programming algorithms use the *Bellman equations* to define iterative algorithms for both policy evaluation and control. Recall the *Bellman equation* for value function v and action-value function q:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_\pi(s') \right], \quad \forall\ s \in \mathcal{S} \tag{1}$$

$$= \mathbb{E}\left[R_{t+1} + \gamma\, v_\pi(S_{t+1}) | S_t = s \right] \tag{2}$$

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right] \tag{3}$$

$$= \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a \right] \tag{4}$$

where $v_\pi : \mathcal{S} \to \mathbb{R}$ is value function and $q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the action-value function. $r(s, a) := \mathbb{E}_\pi\left[R_{t+1} | S_t = s, A_t = a \right]$ and $p(s', r|s, a)$ is the dynamic function of the MDP. $p(s'|s, a)$ is the state transition probability. The optimal value v_* and q_* also follows the *Bellman optimality equa-*

tion:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right] \tag{5}$$

$$= \max_{a \in \mathcal{A}(s)} \sum_{s'} \sum_{r} p(s', r|s, a) \left[r + \gamma v_*(s')\right] \quad s \in \mathcal{S} \tag{6}$$

$$q_*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a' \in \mathcal{A}(s)} q_*(S_{t+1}, a') \Big| S_t = s, A_t = a\right] \tag{7}$$

$$= \sum_{s'} \sum_{r} p(s', r|s, a) \left[r + \gamma \max_{a' \in \mathcal{A}(s)} q_*(s', a')\right] \quad \forall\ s \in \mathcal{S},\ a \in \mathcal{A}(s) \tag{8}$$

From the Bellman equation, we can see that value function of current state is linearly dependent on the expected rewards and the expected *values of all successor states*. In other words, the Bellman equation decomposes the original problem into a series of ***subproblems*** with the ***same structure***. This **recursive structure** is the basis of efficient algorithms such as dynamic programming. In particular, the value function or the optimal value function provides an efficient data structure that allows us to cache the expected long-term future rewards locally at each iteration. 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.

Classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of **a perfect model** (i.e. know dynamic function p for all s, a, s', r) and because of their great ***computational expense***, but they are still important theoretically. In fact, all of reinforcement learning methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment.

Compared to brute-force search on $|\mathcal{A}|^{|\mathcal{S}|}$ over all possible state-action pairs, dynamic algorithm is *polynomial* in terms of $|\mathcal{A}|$ and $|\mathcal{S}|$, which is quite efficient. Thus, dynamic programming is *exponentially faster* than the brute-force search of the policy space.

Generally, solving an MDP gets harder as the number of states grows. The **curse of dimensionality** says that the size of the state space grows exponentially as the number of state variable increases. Clearly, this would lead to problems if we try to sweep the states to perform policy iteration.

2 Policy evaluation (Prediction)

For given arbitrary policy π and dynamic function p, the task to compute the value function v_π is called **policy evaluation**. From (1), we can define an *iterative algorithm* that update the new value function of a state based on the old value function from the successor states, and the expected immediate rewards, along all the one-step transitions possible under the policy being evaluated.

$$v_{t+1}(s) = \sum_{a} \pi(a|s) \sum_{s'} \sum_{r} p(s', r|s, a) \left[r + \gamma v_t(s')\right], \quad \forall\ s \in \mathcal{S} \tag{9}$$

Note that the update is an affine mapping, $\boldsymbol{v}_{t+1} = \boldsymbol{T}\boldsymbol{v}_t + \boldsymbol{r}$, where $\boldsymbol{T}$ is determined by π and p and $\boldsymbol{r}$ is the expected immediate rewards. We call this kind of operation an *expected update*, because they are based on an expectation over *all possible next states* rather than on a sample next state.

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

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

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 $\Delta < \theta$

Figure 2: Policy evaluation [Sutton and Barto, 2018]

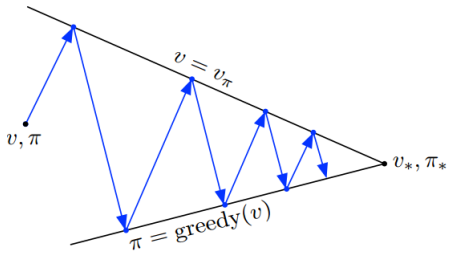

Figure 3: Policy iteration

if $\boldsymbol{r} = \mathbf{0}$, MDP becomes random walk. The iterative algorithm (9) is essentially **Page-Rank algorithm** (9) converges to a fix-point solution $\boldsymbol{v}_t = \boldsymbol{v}_\pi$ which follows the Bellman equation. The bias term $\boldsymbol{r}$ is the expected rewards, which accumulates during the iteration, making sure the value function approximates the goal of reinforcement learning.

The algorithm for in-place policy evaluation algorithm is described in Figure 2. Formally, iterative policy evaluation converges only in the limit, but in practice it must be halted short of this. The pseudocode tests the quantity $\max_{s \in \mathcal{S}} |v_{k+1}(s) - v_k(s)|$ after each sweep and stops when it is sufficiently small. We think of the updates as being done *in a* ***sweep*** through the state space. For the in-place algorithm, the order in which states have their values updated during the sweep has a significant influence on the rate of convergence. We usually have the in-place version in mind when we think of DP algorithms.

3 Policy Iteration (Control)

Our reason for computing the value function for a policy is to help find better policies. For an arbitrary policy π, how to an improved policy π' so that $\pi' \geq \pi$ for every state ? One way to answer this question is to consider selecting a in s and thereafter following the existing policy, π.

The value of this way of behaving is

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

In fact, given policy π and action-value q, we have the following ***policy improvement theorem***

Theorem 3.1 *(**Policy improvement theorem**) Let π and π' be any pair of deterministic policies such that, for all $s \in \mathcal{S}$,*

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{10}$$

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

$$v_{\pi'}(s) \geq v_\pi(s). \tag{11}$$

Moreover, if there is strict inequality of (10) *at any state, then there must be strict inequality of* (11) *at that state.*

Proof: The idea behind the proof of the policy improvement theorem is easy to understand. Starting from (10), we keep expanding the q_π side with (3) and reapplying (10) until we get $v_{\pi'}(s)$:

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

□

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 to a particular action. It is a natural extension to consider changes at all states and to all possible actions, selecting at each state the action that appears best according to $q_\pi(s, a)$. So frr, we only consider the deterministic policy $\pi(s)$, but the proof holds for stochastic policy $\pi(a|s)$.

Given π_t and action-value function q or value function v, a new policy π_{t+1} can be obtained via **greedy algorithm**

$$\begin{aligned} \pi_{t+1}(s) &= \text{argmax}_{a \in \mathcal{A}(s)}\, q_{\pi_t}(s, a), \quad \forall\ s \in \mathcal{S} \\ &= \text{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} \sum_{r} p(s', r|s, a)\left[r + \gamma\, v_{\pi_t}(s')\right] \quad \text{(via (3))} \end{aligned} \tag{12}$$

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 $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s)) \big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
      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

Figure 4: Policy iteration Algorithm

(12) is a greedy algorithm at given state s, the action selected by the new policy π_{t+1} is to maximize the local value function. The greedy policy π_{t+1} takes the action that looks **best** in the short term – *after one step of lookahead* – according to v. (π_{t+1} is deterministic if the max is uniquely obtained; otherwise, it can select each optimal action with equal probability.)

Suppose the new greedy policy, π', is as good as, but not better than, the old policy π. Then $v_{\pi'}(s) = v_\pi(s)$. When the algorithm coverges, the Bellman optimality equation is satisfied. Figure 4 describes the policy iteration algorithm by combining both the policy evaluation in Figure 2 and the policy improvement in (12).

Once a policy, π, has been improved using v to yield a better policy, π', we can then compute v' and improve it again to yield an even better π''. We can thus obtain a sequence of monotonically improving policies and value functions:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

where $\xrightarrow{E}$ denotes a policy evaluation and $\xrightarrow{I}$ denotes a policy improvement. Each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal). Because a finite MDP has only a finite number of policies, this process must converge to an optimal policy and optimal value function in a finite number of iterations. This way of finding an optimal policy is called ***policy iteration***.

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 $V(terminal) = 0$

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

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

Figure 5: Value iteration Algorithm

4 Value iteration

One drawback to policy iteration is that each of its iterations involves *policy evaluation*, which may itself be a protracted iterative computation requiring **multiple sweeps through the state set**. In fact, policy evaluation can be stopped after just one sweep (one update of each state). This algorithm is called ***value iteration***.

$$v_{t+1}(s) = \max_{a \in \mathcal{A}(s)} \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a) v_t(s') \right] \tag{13}$$

$$= \max_{a \in \mathcal{A}(s)} \sum_{s'} \sum_{r} p(s', r|s, a) \left[r + \gamma v_t(s') \right] \quad s \in \mathcal{S} \tag{14}$$

Compare (14) and (12), we can see that instead of computing v_π from old policy π to update π', which from (9) requires averaging over all possible actions, we can directly use the old value v_t to update new value v_{t+1}. Similar to the convergence proof for policy iteration, 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_*, i.e. the Bellman optimality equation. Given v_*, π_* can be obtained via greedy algorithm in (12).

Another way of understanding value iteration is by reference to the Bellman optimality equation (5) and (6). Note that value iteration is obtained simply by turning the Bellman optimality equation into an update rule.

Finally, let us consider how value iteration terminates. Like policy evaluation, value iteration formally requires an infinite number of iterations to converge exactly to v_*. In practice, we stop once the value function changes by only a small amount in a sweep. The Figure 5 shows a complete algorithm with this kind of termination condition.

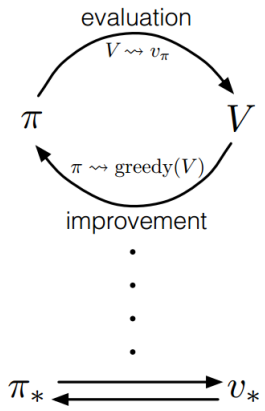

Figure 6: Generalized Policy iteration

5 Generalized Policy Iteration

Policy iteration consists of two simultaneous, interacting processes, 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). In policy iteration, these two processes alternate, each completing before the other begins, but this is not really necessary. In value iteration, for example, only a single iteration of policy evaluation is performed in between each policy improvement. In asynchronous DP methods, the evaluation and improvement processes are interleaved at an even finer grain.

We use the term **generalized policy iteration (GPI)** to refer to the general idea of letting policy-evaluation and policyimprovement processes interact, independent of the granularity and other details of the two processes. 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, as suggested by the diagram to the right. If both the evaluation process and the improvement process stabilize, that is, no longer produce changes, then the value function and policy must be optimal. Figure 6 shows this idea.

The evaluation and improvement processes in GPI can be viewed as both *competing* and *cooperating*. They compete in the sense that they pull in opposing directions. Making the policy greedy with respect to the value function typically makes the value function incorrect for the changed policy, and making the value function consistent with the policy typically causes that 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.

Note that this *Generalized Policy Iteration* is an iterative algorithm with two steps: **policy evaluation** and **control**, which is similar to *Expectation-Maximization (**EM**) algorithm* in statistics/supervised learning. EM algorithm is an iterative algorithm with two steps: *expectation* or **inference** and *maximization* or **learning**. In fact, the control step in GPI is equivalent to the learning step in EM, since both involves finding optimal model (policy vs. model parameter) that maximizing the objective function (expected value vs. log-likelihood function). Similarly, the policy evaluation step in GPI is equivalent to the expectation/inference step in EM, since both involves es-

timating the statistics via taking expectation (value function vs. complete log-likelihood function). Similar to EM algorithm, policy iteration provides a way of heuristic search for optimal policy and optimal value simultaneously.

References

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.