Reading of theoretical aspect of reinforcement learning – Chapter 4

Zhe Rao

- 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)
  - Starting with this chapter, we usually assume that the environment is a finite MDP.
  - 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 can easily obtain optimal policies once we have found the optimal value function, v* or q*, which satisfy the Bellman optimality equations

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_*(s')\Big], \qquad (4.1) \end{aligned}$$

or

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big], \qquad (4.2) \end{aligned}$$

    - ✧ DP algorithms are obtained by turning Bellman equations such as these into assignments, that is, into update rules for improving approximations of the desired value functions

1. Policy Evaluation (Prediction)

- First, we consider how to compute the state-value function v_π for an arbitrary policy π.
  - This is called ***policy evaluation*** in the DP literature.
  - We also refer to it as the ***prediction problem***

- Recall that

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] && \text{(from (3.9))} \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] && (4.3) \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big], && (4.4) \end{aligned}$$

  - Where $\pi(a|s)$ is the probability of taking action a in state s under policy π, and the expectations are subscribed by π to indicate that they are conditional on π being followed

- Iterative solution methods are most suitable.
  - Consider a sequence of approximate value functions $v_0, v_1, v_2, \ldots$ each mapping S^+ to real numbers.

- The initial approximation, v_0 is chosen arbitrarily, and each successive approximation is obtained by using the Bellman equation for v_π as an update rule

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

  - For all $s \in S$.
- Clearly, $v_k = v_\pi$ is a fixed point for this update rule because the Bellman equation for v_π assures us of equality in this case.
- This algorithm is called ***iterative policy evaluation***.

- 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 from the old values of the successor state of s, and the expected immediate rewards, along all the one-step transitions possible under the policy being evaluated.
  - We call this kind of operation an ***expected update***.
  - Each iteration of iterative policy evaluation updates the value of every state once to produce the new approximate value function v_{k+1}

- To write a sequential computer program to implement iterative policy evaluation is as follows

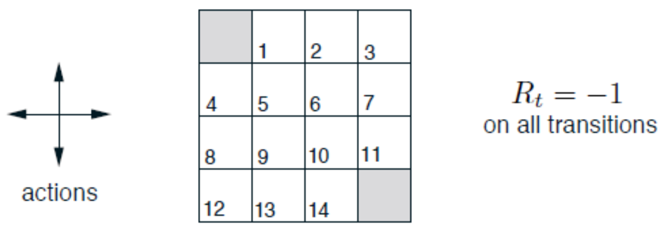
Iterative Policy Evaluation, for estimating $V \approx v_\pi$

```
Input $\pi$, 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 $\Delta < \theta$
```

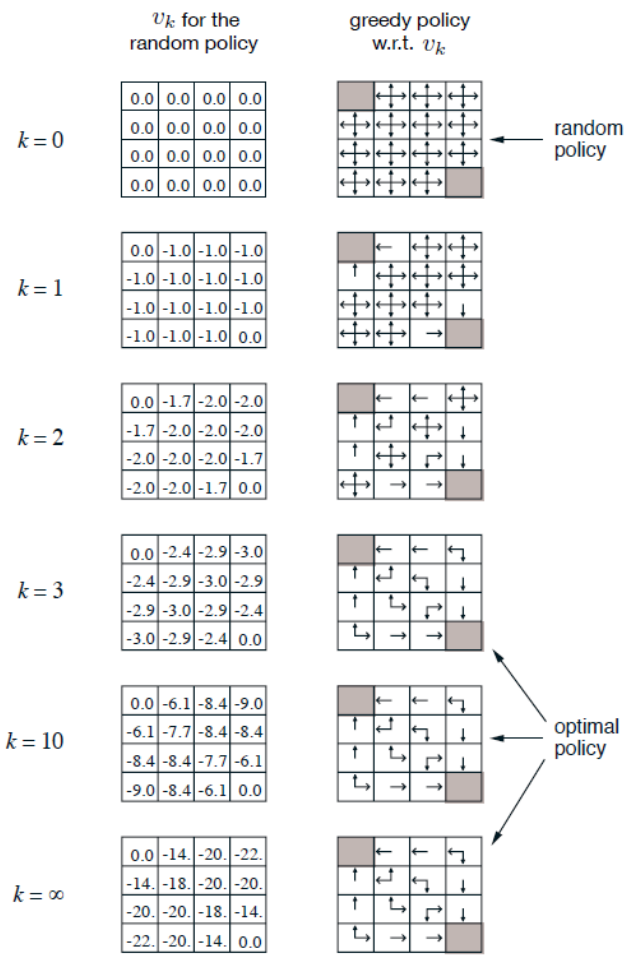
  - You need to use two arrays, one for the old values, $v_k(s)$, and one for the new values, $v_{k+1}(s)$
  - Alternatively, you could use one array and update the value "in place", that is, with each new value immediately overwriting the old one.
    - It usually converges faster than the two-array version, as you might expect, because it uses new data as soon as they are available

Example 4.1



- The nonterminal states are $S = \{1,2,\dots,14\}$.
  - There are four actions possible in each state, $A = \{up,\ down,\ right,\ left\}$, which deterministically cause the corresponding state transitions, expect that actions that would take the agent off the grid in fact leave the state unchanged.
    - Thus, for instance, $p(6,-1|5,\ right) = 1$, $p(7,-1|7,\ right) = 1$, and $p(10,\ r|5,\ right) = 0$
  - This is an undiscounted, episodic task
  - The reward is -1 on all transitions until the terminal state is reached.
  - The terminal state is shaded in the figure (although it is shown in two places, it is formally one state).
  - The expected reward function is thus $r(s,a,s') = -1$.

- Suppose the agent follows the equiprobable random policy (all actions equally likely).
  - The left side of the following figure shows the sequence of value functions $\{v_k\}$ computed by iterative policy evaluation.
  - The final estimate is in fact v_π, which in this case gives for each state the negation of the expected number of steps from that state until termination.



0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

2. Policy Improvement

- Our reason for computing the value function for a policy is to help find better policies.
  - Suppose that we have determined the value function v_π for an arbitrary deterministic policy π.
  - For some state s we would like to know whether or not we should change the policy to deterministically choose an action $a \neq \pi(s)$.
  - We know how good it is to follow the current policy from s – that is $v_\pi(s)$ – but would it be better or worse to change to the new policy?
  - Our 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

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

- The key criterion is whether this is greater than or less than $v_\pi(s)$.
  - ➢ If this 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 be a better one overall

- That this is true is a special case of a general result called the ***policy improvement theorem***
  - ➢ Let π and π' be any pair of deterministic policies such that for all state

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{4.7}$$

  - ➢ Then the policy π' must be as good, or better than, π.
  - ➢ That is, it must obtain greater or equal expected return from all states s

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

- The idea behind the proof of the policy improvement theorem is easy to understand.
  - ➢ Starting from (4.7), we keep expanding the q_π side with (4.6) and reapplying (4.7) until we get $v_{\pi'}(s)$

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] && \text{(by (4.6))} \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] && \text{(by (4.7))} \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\ &= \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s\big] \\ &\leq \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s\big] \\ &\vdots \\ &\leq \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s\big] \\ &= v_{\pi'}(s). \end{aligned}$$

- 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.
  - ➢ It is a natural extension to consider changes at all states, selectin at each state the action that appears best according to $q_\pi(s,a)$.
  - ➢ In other words, to consider the new *greedy* policy, π', given by

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

- The greedy policy takes the action that looks best in the short term – after one step of lookahead – according to v_π.

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

- Suppose the new greedy policy, π', is as good as, but not better than, the old policy π.
  - Then $v_\pi = v_{\pi'}$, it follows that for all state s

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

  - This is the same as the Bellman optimality equation, and therefore, $v_{\pi'}$ must be v^*, and both π and π' must be optimal policies.
  - Policy improvement thus must give us a strictly better policy except when the original policy is already optimal.

- So far in this section we have considered the special case of deterministic policies.
  - In the general case, a stochastic policy π specifies probabilities, $\pi(a|s)$, for taking each action, a, in each state, s.
  - All the ideas of this section extend easily to stochastic policies.
  - In addition, if there are ties in policy improvement steps – that is, if there are several actions at which the maximum is achieved – then in the stochastic case we need not to select a single action from among them.
    - Instead, each maximizing action can be given a portion of the probability of being selected in the new greedy policy.
  - Any apportioning scheme is allowed as long as all the submaximal actions are given zero probability.

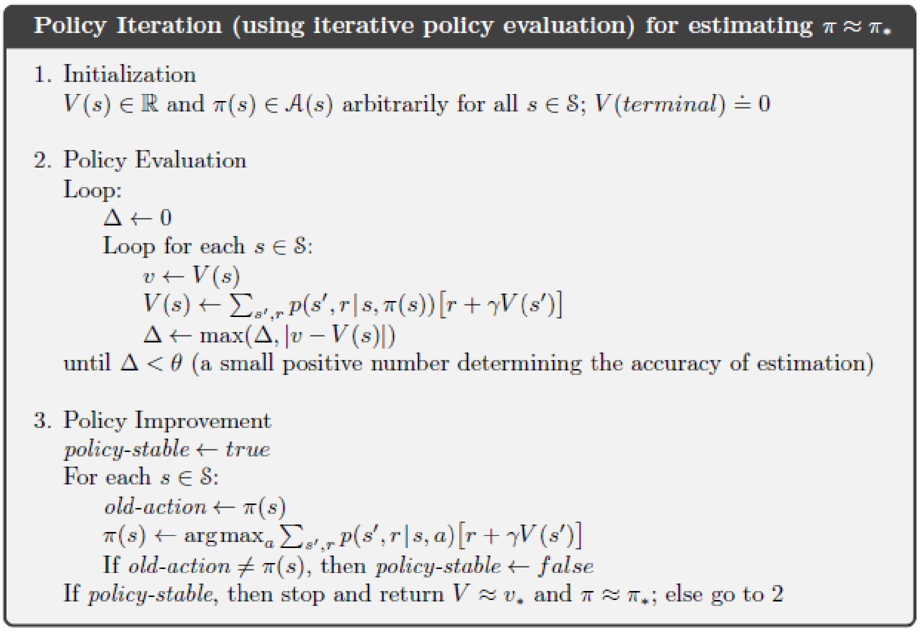
3. *Policy Iteration*

- Once a policy, π, has been improved using v_π to yield a better policy, π', we can then compute $v_{\pi'}$ and improve it again to yield an even better π''.
  - We can thus obtain a sequence of monotonically improving policies and value functions:

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

  - Where → link π_i and v_{π_i} denotes a policy ***evaluation***, and the other arrow 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 deterministic policies, this process must converge to an optimal policy and the optimal value function ins a finite number of iterations.

- This way of finding an optimal policy is called ***policy iteration***.
  - ➢ A complete algorithm is given in the box below



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

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
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $old\text{-}action \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
       If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

Example 4.2: Jack's Car Rental

- Jack manages two locations for a nationwide car rental company
  - ➢ Each day, some number of customers arrive at each location to rent cars.
    - ✧ If Jack has a car available, he rents it out and is credited \$10 by the national company.
    - ✧ If he is out of cars at that location, then the business is lost
    - ✧ Cars become available for renting the day after they are returned.
  - ➢ To help ensure that cars are available when they are needed, Jack can move them between the two locations overnight, at a cost of \$2 per car moved.
  - ➢ We assume that the number of cars requested and returned at each location are Poisson random variables, meaning that the probability that the number is n is $\frac{\lambda^n}{n!}e^{-\lambda}$, where λ is the expected number.
    - ✧ Suppose λ is 3 and 4 for rental requests at the first and second locations and 3 and 2 for returns.
  - ➢ To simplify the problem slightly, we assume that there can be no more than 20 cars at each location (any additional cars are returned to the nationwide company, and thus disappear from the problem) and a maximum of five cars can be moved from one location to the other in one night.
  - ➢ We take the discount rate of 0.9 and formulate this as a continuing finite MDP, where the time steps are days, the state is the number of cars at each location at the end of the day, and the actions are the net number of cars moved between the two locations overnight.

- The following figure shows the sequence of policies found by policy iteration staring from the policy that never moves any cars

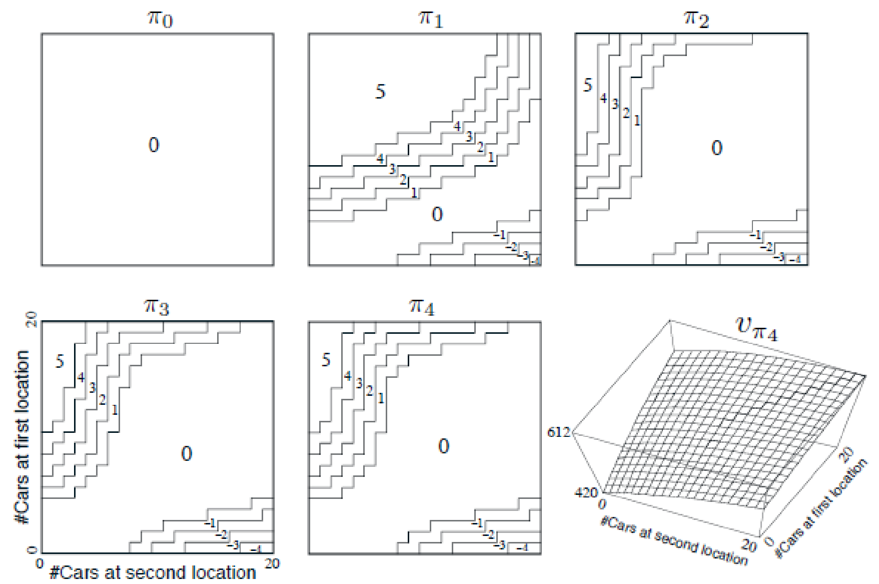


Figure 4.2: The sequence of policies found by policy iteration on Jack's car rental problem, and the final state-value function. The first five diagrams show, for each number of cars at each location at the end of the day, the number of cars to be moved from the first location to the second (negative numbers indicate transfers from the second location to the first). Each successive policy is a strict improvement over the previous policy, and the last policy is optimal. ■

- Policy iteration often converges in surprisingly few iterations.
  - ➢ The bottom-left diagram of the figure shows the value function for the equiprobable random policy, and the bottom-right diagram shows a greedy policy for this value function.
  - ➢ The policy improvement theorem assures us that these policies are better than the original random policy.
  - ➢ In this case, however, these policies are not just better, but optimal, proceeding to the terminal state in the minimum number of steps. In this example, policy iteration would find the optimal policy after just one iteration.

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.
  - ➢ 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?
    - ✧ The example in Figure 4.1 suggests that it may be possible to truncate policy evaluation.
    - ✧ In that example, policy evaluation iterations beyond the first three have no effect on the corresponding greedy policy

- In fact, the policy evaluation step of policy iteration 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***.
  - It can be written as a particularly simple update operation that combines the policy improvement and truncated policy evaluation steps.

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

- 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_*

- Another way of understanding value iteration is by reference to the Bellman optimality equation.
  - Note that value iteration is obtained simply by turning the Bellman optimality equation into an update rule.
  - Also note how the value iteration update is identical to the policy evaluation update (4.5) except that it requires the maximum to be taken over all actions

- Finally, let use 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 box below shows a complete algorithm with this kind of termination condition.

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

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

- 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.
  - In general, the entire class of truncated policy iteration algorithms can be thought of as sequences of sweeps, some of which use policy evaluation updates and some of which use value iteration updates.
  - Because the max operation in (4.10) is the only difference between these updates, this just means that the max operation is added to some sweeps of policy evaluation.

✧ All of these algorithms converge to an optimal policy for discounted finite MDPs

Example 4.3: Gambler's Problem

- A gambler has the opportunity to make bets on the outcomes of a sequence of coin flips.
  - If the coin comes up heads, he wins as many dollars as he has staked on that flip; if it is tails, he loses his stake.
  - The game ends when the gambler wins by reaching his goal of $100, or loses by running out of money.
  - On each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars.

- This problem can be formulated as an undiscounted, episodic, finite MDP.
  - The state is the gambler's capital, $s \in \{1, 2, \dots, 99\}$ and the actions are stakes, $a \in \{0, 1, \dots, \min(s, 100 - s)\}$
  - The reward is zero on all transitions except those on which the gambler reaches his goal, when it is +1
  - The state-value function then gives the probability of winning from each state
  - A policy is a mapping from levels of capital to stakes.
  - The optimal policy maximizes the probability of reaching the goal

- Let p_h denote the probability of the coin coming up heads.
  - If p_h is known, then the entire problem is known and it can be solved, for instance, by value iteration.
  - The following figure shows the change in the value function over successive sweeps of value iteration, and the final policy found, for the case of $p_h = 0.4$.
  - This policy is optimal, but not unique. In fact, there is a whole family of optimal policies, all corresponding to ties for the argmax action selection with respect to the optimal value function.

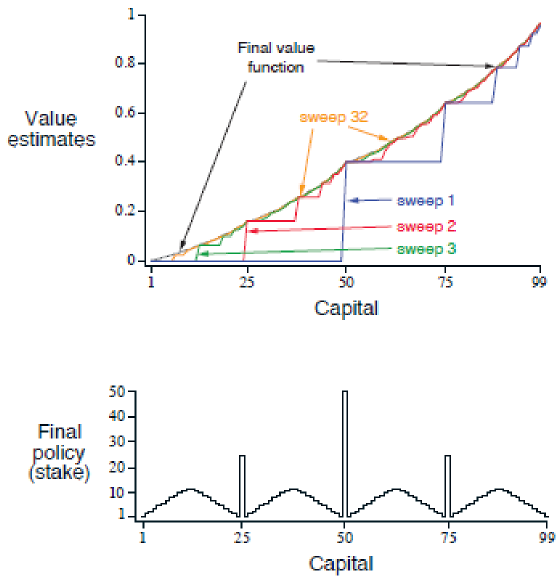


Figure 4.3: The solution to the gambler's problem for $p_h = 0.4$. The upper graph shows the value function found by successive sweeps of value iteration. The lower graph shows the final policy. ■

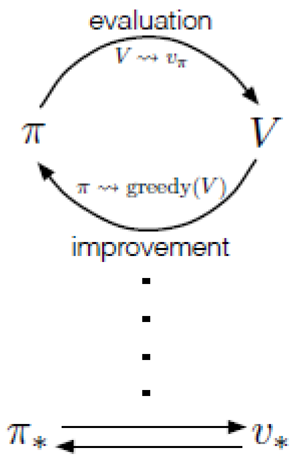
5. Asynchronous Dynamic Programming

- A major drawback to the DP methods that we have discussed so far is that they involve operations over the entire state set of the MDP, that is, 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 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 whatever values of other states happen to be available.
    - The value of some states may be updated several times before the value 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 DP algorithms allow great flexibility in selecting states to update

6. 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
  - As long as both processes continue to update all states, the ultimate result is typically the same – convergence to the optimal value function and an optimal policy

- We use the term ***generalized policy iteration*** (GPI) to refer to the general idea of letting policy-evaluation and policy-improvement process interact, independent of the granularity and other details of the two processes.
  - All reinforcement learning methods 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 below.
  - If both the evaluation process and the improvement process stabilize, the value function and policy must be optimal.
  - The value function stabilizes only when it is consistent with the current policy, and the policy stabilizes only when it is greedy with respect to the current value function.
  - This implies that the Bellman optimality equation holds, and thus that the policy and the value function are optimal.


evaluation
$V \rightsquigarrow v_\pi$
π
V
$\pi \rightsquigarrow \text{greedy}(V)$
improvement
π_*
v_*

- 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
    - ✧ 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

7. Efficiency of Dynamic Programming

- 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, the, 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 n and k denote the number of states and actions, this means that a DP method takes a number of computational operations that is less than some polynomial function of n and k.
  - ➢ A DP method is guaranteed to find an optimal policy in polynomial time even though the total number of (deterministic policies) is k^n

- DP is sometimes thought to be of limited applicability because of the ***curse of dimensionality***
  - ➢ The fact that the number of states often grows exponentially with the number of states variables.