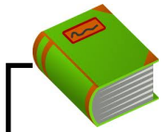
MDPs: value iteration

- If we are given a policy π, we now know how to compute its value $V_\pi(s_{\text{start}})$. So now, we could just enumerate all the policies, compute the value of each one, and take the best policy, but the number of policies is exponential in the number of states (A^S to be exact), so we need something a bit more clever.
- We will now introduce value iteration, which is an algorithm for finding the best policy. While evaluating a given policy and finding the best policy might seem very different, it turns out that value iteration will look a lot like policy evaluation.

Optimal value and policy

Goal: try to get directly at maximum expected utility

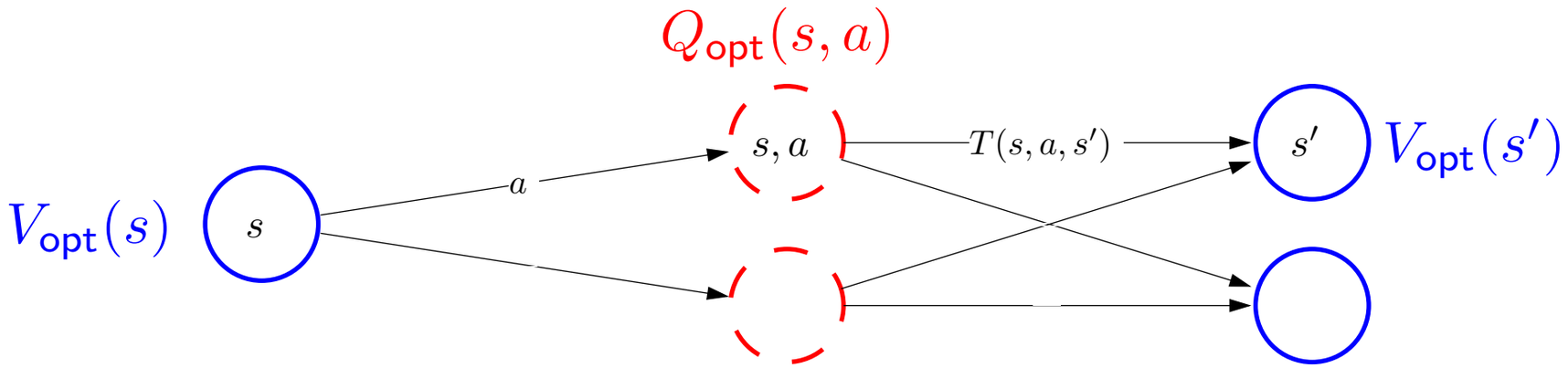


Definition: optimal value

The **optimal value** $V_{\text{opt}}(s)$ is the maximum value attained by any policy.

- We will write down a bunch of recurrences which look exactly like policy evaluation, but instead of having V_π and Q_π with respect to a fixed policy π, we will have V_{opt} and Q_{opt}, which are with respect to the optimal policy.

Optimal values and Q-values



Optimal value if take action a in state s:

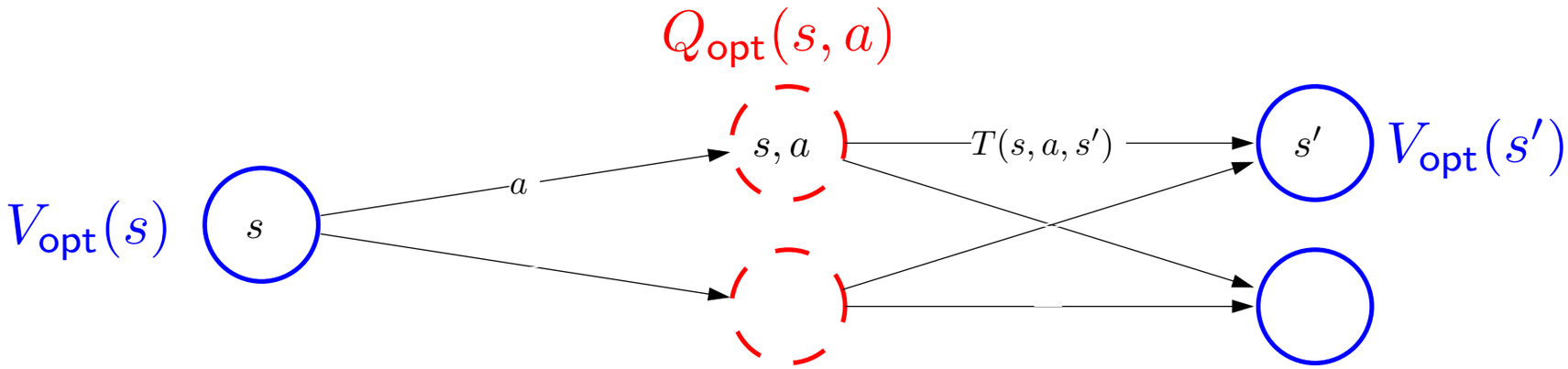
$$Q_{\text{opt}}(s,a) = \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}(s')].$$

Optimal value from state s:

$$V_{\text{opt}}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s,a) & \text{otherwise.} \end{cases}$$

- The recurrences for V_{opt} and Q_{opt} are identical to the ones for policy evaluation with one difference: in computing V_{opt}, instead of taking the action from the fixed policy π, we take the best action, the one that results in the largest $Q_{\text{opt}}(s, a)$.

Optimal policies



Given Q_{opt}, read off the optimal policy:

$$\pi_{\text{opt}}(s) = \arg\max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s,a)$$

- So far, we have focused on computing the value of the optimal policy, but what is the actual policy? It turns out that this is pretty easy to compute.
- Suppose you're at a state s. $Q_{\text{opt}}(s, a)$ tells you the value of taking action a from state s. So the optimal action is simply to take the action a with the largest value of $Q_{\text{opt}}(s, a)$.

Value iteration

Algorithm: value iteration [Bellman, 1957]

Initialize $V_{\text{opt}}^{(0)}(s) \leftarrow 0$ for all states s.

For iteration $t = 1, \ldots, t_{\text{VI}}$:

For each state s:

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \underbrace{\sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')]}_{Q_{\text{opt}}^{(t-1)}(s,a)}$$

Time: $O(t_{\text{VI}} SAS')$

[semi-live solution]

- By now, you should be able to go from recurrences to algorithms easily. Following the recipe, we simply iterate some number of iterations, go through each state s and then replace the equality in the recurrence with the assignment operator.
- Value iteration is also guaranteed to converge to the optimal value.
- What about the optimal policy? We get it as a byproduct. The optimal value $V_{\text{opt}}(s)$ is computed by taking a max over actions. If we take the argmax, then we get the optimal policy $\pi_{\text{opt}}(s)$.

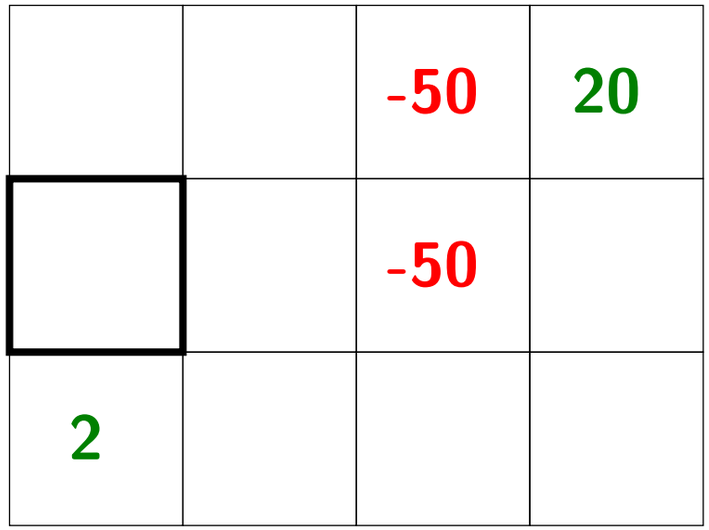
Value iteration: dice game

s	end	in	
$V_{\text{opt}}^{(t)}$	0.00	12.00	($t = 100$ iterations)
$\pi_{\text{opt}}(s)$	-	stay	

- Let us demonstrate value iteration on the dice game. Initially, the optimal policy is "quit", but as we run value iteration longer, it switches to "stay".

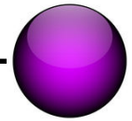
Value iteration: volcano crossing

Run (or press ctrl-enter)



- As another example, consider the volcano crossing. Initially, the optimal policy and value correspond to going to the safe and boring 2. But as you increase `numIters`, notice how the value of the far away 20 propagates across the grid to the starting point.
- To see this propagation even more clearly, set `slipProb` to 0.

Convergence



Theorem: convergence

Suppose either

- discount $\gamma < 1$, or
- MDP graph is acyclic.

Then value iteration converges to the correct answer.

Example: non-convergence

discount $\gamma = 1$, zero rewards

- Let us state more formally the conditions under which any of these algorithms that we talked about will work. A sufficient condition is that either the discount γ must be strictly less than 1 or the MDP graph is acyclic.
- We can reinterpret the discount $\gamma < 1$ condition as introducing a new transition from each state to a special end state with probability $(1-\gamma)$, multiplying all the other transition probabilities by γ, and setting the discount to 1. The interpretation is that with probability $1-\gamma$, the MDP terminates at any state.
- In this view, we just need that a sampled path be finite with probability 1.
- We won't prove this theorem, but will instead give a counterexample to show that things can go badly if we have a cyclic graph and $\gamma = 1$. In the graph, whatever we initialize value iteration, value iteration will terminate immediately with the same value. In some sense, this isn't really the fault of value iteration, but it's because all paths are of infinite length. In some sense, if you were to simulate from this MDP, you would never terminate, so we would never find out what your utility was at the end.

Summary of algorithms

- Policy evaluation: (MDP, π) $\rightarrow V_\pi$

- Value iteration: MDP $\rightarrow (Q_{\text{opt}}, \pi_{\text{opt}})$

Unifying idea

Algorithms:

- Search DP computes FutureCost(s)
- Policy evaluation computes policy value $V_\pi(s)$
- Value iteration computes optimal value $V_{\text{opt}}(s)$

Recipe:

- Write down recurrence (e.g., $V_\pi(s) = \cdots V_\pi(s') \cdots$)
- Turn into iterative algorithm (replace mathematical equality with assignment operator)

- There are two key ideas in this lecture. First, the policy π, value V_π, and Q-value Q_π are the three key quantities of MDPs, and they are related via a number of recurrences which can be easily gotten by just thinking about their interpretations.
- Second, given recurrences that depend on each other for the values you're trying to compute, it's easy to turn these recurrences into algorithms that iterate between those recurrences until convergence.