1 Learning Goals

- Compute the optimal action for a state given the true utilities of all the states.
- Trace the execution of the value iteration algorithm. Calculate the true utilities of a state for the current iteration given the true utilities of all the states for the previous iteration.

2 Algorithms for finding the optimal policy

2.1 Optimal policies and the utilities of states

Suppose that we enter state s and follow the optimal policy from s onward, what is our total expected utility? This is a way to measure the "true utility" of each state s, denoted by $V(s)$.

$V(s)$ and $R(s)$ are very different. What is the difference?

- $R(s)$ is one-time reward of entering state s
- $V(s)$ is long-term total discounted reward of starting from state s and following the optimal policy.

Figure 1 shows $V(s)$ for every state (with discount factor $\gamma = 1$ and $R(s) = -0.04$). The utilities are higher for states closer to the +1 exit because fewer steps are required to reach the exit.

	1	2	3	4
1	0.705	0.655	0.611	0.388
2	0.762	X	0.660	-1
3	0.812	0.868	0.918	+1

Figure 1: The true utilities $V(s)$ for $\gamma = 1$ and $R(s) = -0.04, \forall s \neq s_{24}, s \neq s_{34}$.

Given $V(s)$, it's straightforward to determine the optimal action in each state.

What is my expected utility if I'm in state s and take action a?

$$V(s, a) = \sum_{s'} P(s'|s, a)V(s')$$

Given this, I would like to choose the action that maximizes my expected utility.

$$\pi^*(s) = \arg\max_a V(s,a).$$

What is the optimal policy in s_{13}?

- $V(s_{13}, \text{down})$ $= 0.8 * 0.660 + 0.1 * 0.655 + 0.1 * 0.388 = 0.6323$
- $V(s_{13}, \text{left})$ $= 0.8 * 0.655 + 0.1 * 0.660 + 0.1 * 0.611 = 0.6511$
- $V(s_{13}, \text{right})$ $= 0.8 * 0.388 + 0.1 * 0.611 + 0.1 * 0.660 = 0.4375$
- $V(s_{13}, \text{up})$ $= 0.8 * 0.611 + 0.1 * 0.655 + 0.1 * 0.388 = 0.5931$

Therefore, $\pi^*(s_{13})$ $=$ left.

Now the question is, how do we determine $V(s)$?

2.2 Value iteration

Basic idea:

- Calculate the true utility $V(s)$ of each state s.
- Choose the optimal action in each state based on $V(s)$.

The true utilities $V(s)$ are the unique solutions to the Bellman equations.

$$V(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V(s')$$

- $R(s)$ is the one-time immediate reward for entering state s.
- $\gamma \max_a \sum_{s'} P(s'|s,a)V(s')$ is the discounted expected utility of the next state, assuming that the agent chooses the optimal action.

The Bellman equation for $V(s_{11})$:

$$\begin{aligned} V(s_{11}) = -0.04 + \gamma \max[&0.8 * V(s_{12}) + 0.1 * V(s_{21}) + 0.1 * V(s_{11}), \\ &0.9 * V(s_{11}) + 0.1 * V(s_{12}), \\ &0.9 * V(s_{11}) + 0.1 * V(s_{21}), \\ &0.8 * V(s_{21}) + 0.1 * V(s_{12}) + 0.1 * V(s_{11})]. \end{aligned}$$

For our example, there are 9 Bellman equations, one for each state. We can solve these 9 equations to find the 9 unknowns $V(s)$.

Can we solve this system of equations using an efficient algorithm?

- No. These equations are non-linear since "max" is non-linear.
- We can solve linear equations efficiently using linear algebra techniques.

Instead, we will solve for $V(s)$ using an iterative approach.

1. Start with arbitrary initial values for the true utilities.

   Let $V_i(s)$ be the true utility for state s in the ith iteration.
2. At the ith iteration, compute all the $V_{i+1}(s)$ using the following update rule.

$$V_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_i(s')$$

3. Terminate when the maximum change from $V_i(s)$ to $V_{i+1}(s)$ for all s is small enough.

If we apply the Bellman update infinitely often, we are guaranteed to converge to the optimal $V(s)$.

A few notes about applying the update rule:

- We apply the update rule from right to left. Plug in the old utility values on the right. The result from the right becomes the new utility value.
- We will update all the true utility values simultaneously. This means that, we will use the utility values from the previous iteration to calculate the utility values for the current iteration, and then we will update all the utility value at once.

Let's apply the value iteration algorithm.

Assume that the discount factor $\gamma = 1$ and $R(s) = -0.04, \forall s \neq s_{24}, s \neq s_{34}$.

$V_0(s)$:

	1	2	3	4
1	0	0	0	0
2	0	X	0	-1
3	0	0	0	+1

$V_1(s_{23}) =$

$V_1(s_{33}) =$

$$\begin{aligned}
V_1(s_{23}) =& -0.04 + 1 * max[0.8 * 0 + 0.1 * (-1) + 0.1 * 0,\\
&\qquad 0.8 * (-1) + 0.1 * 0 + 0.1 * 0,\\
&\qquad 0.8 * 0 + 0.1 * (-1) + 0.1 * 0,\\
&\qquad 0.8 * 0 + 0.1 * 0 + 0.1 * 0]\\
=& -0.04 + 0 = -0.04\\
V_1(s_{33}) =& -0.04 + 1 * max[0.8 * 1 + 0.1 * 0 + 0.1 * 0,\\
&\qquad 0.8 * 0 + 0.1 * 1 + 0.1 * 0,\\
&\qquad 0.8 * 0 + 0.1 * 0 + 0.1 * 0,\\
&\qquad 0.8 * 0 + 0.1 * 1 + 0.1 * 0]\\
=& -0.04 + 0.8 = 0.76
\end{aligned}$$

$V_1(s)$:

	1	2	3	4
1				
2		X		-1
3				+1

$V_2(s_{33}) =$

$V_2(s_{23}) =$

$V_2(s_{32}) =$

$V_1(s)$:

	1	2	3	4
1	-0.04	-0.04	-0.04	-0.04
2	-0.04	X	-0.04	-1
3	-0.04	-0.04	0.76	+1

$$\begin{aligned}
V_2(s_{33}) =& -0.04 + 1 * max[0.8 * 1 + 0.1 * (-0.04) + 0.1 * 0.76,\\
&\quad 0.8 * 0.76 + 0.1 * 1 + 0.1 * (-0.04),\\
&\quad 0.8 * (-0.04) + 0.1 * (-0.04) + 0.1 * 0.76,\\
&\quad 0.8 * (-0.04) + 0.1 * (-0.04) + 0.1 * 1]\\
=&0.832\\
V_2(s_{23}) =& -0.04 + 1 * max[0.8 * 0.76 + 0.1(-0.04) + 0.1(-1),\\
&\quad 0.8 * (-1) + 0.1 * (-0.04) + 0.1 * 0.76,\\
&\quad 0.8 * 0.76 + 0.1 * (-1) + 0.1 * (-0.04),\\
&\quad 0.8 * (-0.04) + 0.1 * (-0.04) + 0.1 * 0.76]\\
=&0.464\\
V_2(s_{32}) =& -0.04 + 1 * max[0.8 * 0.76 + 0.1(-0.04) + 0.1(-0.04),\\
&\quad 0.8 * (-0.04) + 0.1 * 0.76 + 0.1 * (-0.04),\\
&\quad 0.8 * (-0.04) + 0.1 * 0.76 + 0.1 * (-0.04),\\
&\quad 0.8 * (-0.04) + 0.1 * (-0.04) + 0.1 * (-0.04)]\\
&= 0.56
\end{aligned}$$

$V_2(s)$:

	1	2	3	4
1				
2		X		-1
3				+1

$V_2(s)$:

	1	2	3	4
1	-0.08	-0.08	-0.08	-0.08
2	-0.08	X	0.464	-1
3	-0.08	0.56	0.832	+1

$$V_3(s_{33}) = -0.04 + 0.8 * 1 + 0.1 * 0.464 + 0.1 * 0.832 = 0.890$$
$$V_3(s_{23}) = -0.04 + 0.8 * 0.832 + 0.1(0.464) + 0.1(-1) = 0.572$$
$$V_3(s_{32}) = -0.04 + 0.8 * 0.832 + 0.1(0.56) + 0.1(0.56) = 0.738$$
$$V_3(s_{31}) = -0.04 + 0.8 * 0.56 + 0.1(-0.08) + 0.1(-0.08) = 0.392$$
$$V_3(s_{13}) = -0.04 + 0.8 * 0.464 + 0.1(-0.08) + 0.1(-0.08) = 0.315$$

$V_3(s)$:

	1	2	3	4
1				
2		X		-1
3				+1

$V_3(s)$:

	1	2	3	4
1	-0.12	-0.12	0.315	-0.12
2	-0.12	X	0.572	-1
3	0.392	0.738	0.890	+1

$V_4(s)$:

	1	2	3	4
1				
2		X		-1
3				+1

$V_4(s)$:

	1	2	3	4
1	-0.16	0.188	0.394	0.100
2	0.250	X	0.629	-1
3	0.577	0.819	0.906	+1

$V_5(s)$:

	1	2	3	4
1				
2		X		-1
3				+1

$V_5(s)$:

	1	2	3	4
1	0.162	0.313	0.492	0.185
2	0.472	X	0.648	-1
3	0.698	0.849	0.914	+1

The states accumulate negative rewards until a path is found to s_{34}.

2.3 Policy iteration

This material is for your interest only and will not appear on an exam.

One thing we learned from value-iteration: It's possible to get an optimal policy even when the estimates of the true utilities are inaccurate. If one action is clearly better than all others, then the exact magnitude of the true utilities of the states need not be precise.

Policy iteration algorithm:

1. Begin with some initial policy π_0.
2. Alternative between policy evaluation and policy improvement.
3. Policy evaluation: given a policy π_i, calculate $V_i = V^{\pi_i}$, the utility of each state if the agent follows policy π_i.

   We need to solve a system of linear equations.

   $$V_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))V_i(s')$$

   We can solve this exactly using linear algebra techniques, or we can solve this approximately using value-iteration.
4. Policy improvement: calculate the best policy π_{i+1} based on V_i.
5. Terminate when the policy improvement step yields no change in the utilities.

For a finite state space, there are only finitely many policies. If each iteration yields a better policy, then policy iteration must terminate.

Asynchronous policy iteration

We don't have to update the utility or policy for all states at once.

Asynchronous policy iteration:

- On each iteration, we can pick any subset of states and apply either kind of updating (policy improvement or simplified value iteration) to that subset.
- This is guaranteed to converge to an optimal policy under certain conditions.
- Can design heuristic algorithms – concentrate on updating the values of states that are likely to be reached by a good policy.