Solving for Utility

CS 3600

Intro to Artificial Intelligence

Definitions

Utility (17.2)

S_0=s, S_1, S_2, S_3…

$$U^{\pi}(s) = \mathbb{E}_{S_t \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

Optimal policy (17.4)

$$\pi^*(s) = \arg\max_a \sum_{s'} p(s' \mid a, s) \cdot U^{\pi^*}(s')$$

Bellman equation

If we **know** $U^{\pi^*}(s)$, we can compute the optimal policy. How do we find $U^{\pi^*}(s)$?

Answer: the Bellman equation (17.5)

$$U^{\pi^*}(s) = R(s) + \gamma \max_a \sum_{s'} p(s' \mid a, s) U^{\pi^*}(s')$$

Where did this equation come from?

Bellman proof (1)

Expectation facts

$$\mathbb{E}[A+B] = \mathbb{E}[A] + \mathbb{E}[B]$$
$$\mathbb{E}[c] = c$$

Start with the definition 17.2 and see what we can do

$$U^\pi(s) = \mathbb{E}_{S_t\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(S_t)\right]$$
$$= \mathbb{E}_{S_t\sim\pi}\left[\gamma^0 R(S_0) + \sum_{t=1}^{\infty}\gamma^t R(S_t)\right]$$
$$= R(s) + \boxed{\mathbb{E}_{S_t\sim\pi}\left[\sum_{t=1}^{\infty}\gamma^t R(S_t)\right]}$$

constant
$S_0 = s$

Not *quite* $U^{\pi^*}(s')$

Bellman proof (2)

More expectation facts

$$\mathbb{E}[A] = \mathbb{E}[\mathbb{E}[A \mid B]]$$

$$\mathbb{E}[f(A)] = \sum_a f(A) \cdot p(A = a)$$

S_0=s, S_1, S_2, S_3…

$$U^\pi(s) = R(s) + \mathbb{E}_{S_t \sim \pi}\left[\sum_{t=1}^{\infty} \gamma^t R(S_t)\right]$$

S_0=s, S_1=s', S_2, S_3…

$$= R(s) + \mathbb{E}_{S_t \sim \pi}\left[\mathbb{E}_{S_t \sim \pi}\left[\sum_{t=1}^{\infty} \gamma^t R(S_t) \mid S_1 = s'\right]\right]$$

$$= R(s) + \sum_{s'} p(s' \mid \pi(s), s) \mathbb{E}_{S_t \sim \pi}\left[\sum_{t=1}^{\infty} \gamma^t R(S_t) \mid S_1 = s'\right]$$

Getting closer, but now the index is off

Bellman proof (3)

$$U^\pi(s) = R(s) + \sum_{s'} p(s' \mid \pi(s), s) \mathbb{E}_{S_t \sim \pi}\left[\sum_{t=1}^{\infty} \gamma^t R(S_t) \mid S_1 = s'\right]$$

$$= R(s) + \sum_{s'} p(s' \mid \pi(s), s) \mathbb{E}_{S_t \sim \pi}\left[\gamma \sum_{t=0}^{\infty} \gamma^t R(S_t) \mid S_0 = s'\right]$$

$$= R(s) + \gamma \sum_{s'} p(s' \mid \pi(s), s) \boxed{\mathbb{E}_{S_t \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t) \mid S_0 = s'\right]}$$

Here's U^π(s')!

$$= R(s) + \gamma \sum_{s'} p(s' \mid \pi(s), s) U^\pi(s')$$

Can't compute this if we don't know π, what to do?

Bellman proof (4)

Optimal policy

$$\pi^*(s) = \arg\max_a \sum_{s'} p(s' \mid a, s) \cdot U^{\pi^*}(s')$$

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} p(s' \mid \pi(s), s) U^{\pi}(s')$$

$$U^{\pi^*}(s) = R(s) + \gamma \sum_{s'} p(s' \mid \pi^*(s), s) U^{\pi^*}(s')$$

$$= R(s) + \gamma \max_a \sum_{s'} p(s' \mid a, s) U^{\pi^*}(s')$$

Proof done!

Using Bellman

$$U^{\pi^*}(s) = R(s) + \gamma \max_a \sum_{s'} p(s' \mid a, s) U^{\pi^*}(s')$$

How can we use this to solve for actual values of U(s)?

- This is a system of *non*linear equations (because of max)
- Instead of asserting equality, what if we used it as an **update**?

Bellman **Update**:

$$U^{(i+1)}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} p(s' \mid a, s) U^{(i)}(s')$$

Value Iteration - algorithm

```
def Value_Iteration():
     Initialize U
     for i in iterations:
          Initialize new_U
          for s in states:
               new_U(s) = R(s) + gamma* max(sum(p(s'|a,s)*U(s')))
          U = new_U
          Optionally break once convergence criteria met
```

(figure 17.4 in text)

Iteratively apply the Bellman update until we converge or run out of iterations

Convergence check (not needed for project)

$$\|U^{(i+1)} - U^{(i)}\| < \epsilon(1-\gamma)/\gamma \implies \|U^{(i+1)} - U^{\pi^*}\| < \epsilon$$

Value Iteration - convergence proof (1)

How do we know just applying the Bellman update over and over again will eventually converge to the correct utilities?

Notation

$$U^{(i+1)}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} p(s' \mid a, s) U^{(i)}(s')$$

$$U^{(i+1)} \leftarrow B[U^{(i)}]$$

We call this “applying the Bellman operator”

Definition

$$\left\| U^{(i+1)} - U^{(i)} \right\| = \max_s \left| U^{(i+1)}(s) - U^{(i)}(s) \right|$$

Value Iteration - convergence proof (2)

FACT: The Bellman operator is a **contraction**

$$\|B[U] - B[U']\| \leq \gamma \, \|U - U'\|$$

This means applying the Bellman operator brings any two U's closer together (for hints at why, see exercise 17.6)

Value Iteration - convergence proof (3)

Definition: If B[U] = U, U is called a **fixed point** of the operator B (applying the operator doesn't change anything)

FACT: Any contraction can have at most 1 fixed point

$$\text{Assume } U \neq V \text{ are both fixed points}$$

$$\|B[U] - B[V]\| \leq \gamma\|U - V\|$$

$$\|U - V\| \leq \gamma\|U - V\|$$

$$\implies \|U - V\| = 0$$

$$\implies U = V \quad \text{contradiction!}$$

Value Iteration - convergence proof (4)

So the Bellman operator is a contraction, it can have **at most** one fixed point. How does that help?

The Bellman **equation** tells us that U^{π^*} is a fixed point of the Bellman **operator**!

Bellman Equation

$$U^{\pi^*}(s) = R(s) + \gamma \max_a \sum_{s'} p(s' \mid a, s) U^{\pi^*}(s')$$

Bellman Update

$$U^{(i+1)}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} p(s' \mid a, s) U^{(i)}(s')$$

Value Iteration - example (1)

States $\{s_1, s_2, s_3, s_4\}$

Actions {up, down, left, right}

Transition probability: "0.8 correct, 0.1 perp"
$p(s_4 | R, s_1) = 0.8$, $p(s_1 | R, s_1) = 0.1$

Rewards: "1.0 for s_4, -0.04 for others"

Discount: 0.5

Initial U: 0.1

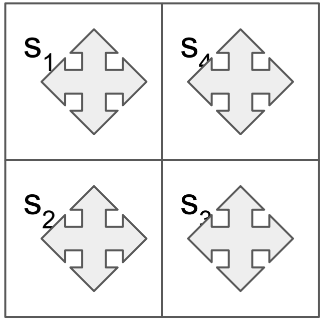
Initial π: any action

s_1 R=-0.04	s_4 R=1.0
s_2 R=-0.04	s_3 R=-0.04

Initial U

s_1 U=.1	s_4 U=.1
s_2 U=.1	s_3 U=.1

Initial π



Value Iteration - example (2)

$$U^{(i+1)}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} p(s' \mid a, s) U^{(i)}(s')$$

s_1 R=-0.04	s_4 R=1.0
s_2 R=-0.04	s_3 R=-0.04

Iteration 1

$U^{(1)}(s_1)$ = -0.04 + (0.5)*max{ ...0.1…} = 0.01

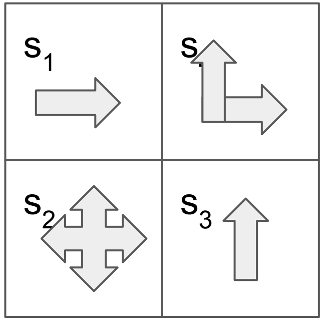
$U^{(1)}(s_2) = U^{(1)}(s_3)$ = 0.01

$U^{(1)}(s_4)$ = 1.0 + (0.5)*max{ ...0.1…} = 1.05

$U^{(1)}$

s_1 U=.01	s_4 U=1.05
s_2 U=.01	s_3 U=.01

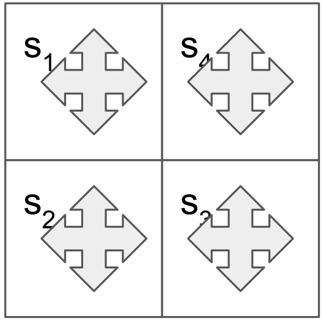
$\pi^{(1)}$



$U^{(0)}$

s_1 U=.1	s_4 U=.1
s_2 U=.1	s_3 U=.1

$\pi^{(0)}$



Value Iteration - example (3)

$$U^{(i+1)}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} p(s' \mid a, s) U^{(i)}(s')$$

Iteration 2

$U^{(2)}(s_1)$ = -0.04 + (0.5)*max{

U: (0.9)(0.01) + (0.1)(1.05)

(0.0)(0.01) + (0.0)(0.01) = 0.114

D: (0.1)(0.01) + (0.1)(1.05) +

(0.8)(0.01) + (0.0)(0.01) = 0.114

L: (0.9)(0.01) + (0.0)(1.05) +

(0.1)(0.01) + (0.0)(0.01) = 0.01

R: (0.1)(0.01) + (0.8)(1.05) +

(0.1)(0.01) + (0.0)(0.01) = 0.8423 }

$U^{(2)}(s_1)$ = -0.04 + (0.5)*(0.8423) = 0.381

s_1 R=-0.04	s_4 R=1.0
s_2 R=-0.04	s_3 R=-0.04

$U^{(1)}$

s_1 U=.01	s_4 U=1.05
s_2 U=.01	s_3 U=.01

Value Iteration - example (4)

$$U^{(i+1)}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} p(s' \mid a, s) U^{(i)}(s')$$

Iteration 2

$U^{(2)}(s_1) = 0.381$

$U^{(2)}(s_2) = -0.04 + (0.5)(0.01) = -0.035$

$U^{(2)}(s_3) = 0.381$ (symmetric w/ s_1)

$U^{(2)}(s_4) = 1.0 + (0.5)\max\{$

U: $(0.1)(0.01) + (0.9)(1.05) = 0.946$

D: low

L: low

R: 0.946 (symmetric w/ U) $\} = 1.0+0.473$

$U^{(2)}(s_4) = 1.473$

s_1 R=-0.04	s_4 R=1.0
s_2 R=-0.04	s_3 R=-0.04

$U^{(2)}$

s_1 U=.381	s_4 U=1.47
s_2 U=-.035	s_3 U=.381

$U^{(1)}$

s_1 U=.01	s_4 U=1.05
s_2 U=.01	s_3 U=.01

Value Iteration - notes

We found the correct policy after just two iterations!

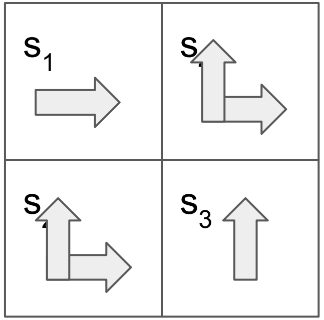
But our utility function hadn't converged yet…

Notes

- Policy tends to converge faster than utility
- Each iteration only updates utility from states "one hop away:" may take a long time to propagate from goal
- If we could "fix" the policy, the max operator would go away, and the Bellman update would be linear

s_1 R=-0.04	s_4 R=1.0
s_2 R=-0.04	s_3 R=-0.04

$\pi^{(2)}$



Policy Iteration

If the policy is fixed, the Bellman equation becomes

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} p(s' \mid \pi(s), s) U^{\pi}(s')$$

We can use this to implement `Policy_Eval()` as either a simplified version of Value Iteration, or by solving a system of linear equations

```
def Policy_Iteration():
     Initialize U, policy
     for i in iterations:
          new_U = Policy_Eval(policy, U)
          for s in states:
               a’ = argmax(sum( p(s’|a,s)*new_U[s’]))
               if a’ is better than old policy with new_U:
                    new_policy[s] = a’
               else:
                    new_policy[s] = policy[s]
          if policy[s] = new_policy[s] for all s:
               break
          else
               policy = new_policy
```

Summary and preview

Wrapping up

- The Bellman equation gives us a way to compute the **utility** of the optimal policy, which in turn gives us a way to **find** the optimal policy
- Value Iteration is an iterative method that is **guaranteed** to converge to the optimal **utility** values
- Policy Iteration can potentially **speed up** this process, by keeping the policy fixed and using a simplified form of Bellman

Next time: Examples and Reinforcement Learning