Policy Iteration (Ch. 17.3)

Image credit: P. Abbeel and D. Klein

Announcements

HW 3 due Sunday

Markov Decision Process

 = +50 (end)

 = -50 (end)

All other = -1 (i.e. -1 for movement)

Goal: maximize score before reaching end

Markov Decision Process

When the robot tries to move, 80% of the time it ends up where it wants to go

10% it will end up 90 degrees off

Wall = no move

MDP Utilities

$S = [s_0, s_1, s_2, s_3, ...]$, $S' = [s_0', s_1', s_2', s_3', ...]$
Suppose you prefer S over S' and $s_0 = s_0'$

... If you can then conclude that you would prefer $[s_1, s_2, s_3, ...]$ over $[s_1', s_2', s_3', ...]$

We call this a stationary preference
(and it has some large implications)

MDP Utilities

If you have a stationary preference, then there are only two valid utility functions:

Additive:

$$U(s_0, s_1, s_2, ...) = R(s_0) + R(s_1) + R(s_2) + ...$$

Discounted:

$$U(s_0, s_1, s_2, ...) = R(s_0) + \gamma \cdot R(s_1) + \gamma^2 \cdot R(s_2) + ...$$

... where $0<\gamma<1$

Value Iteration Convergence

You simply repeat this process until the numbers “converge” (i.e. stop changing much)

In fact, it is both guaranteed to converge always and within a bounded amount

It has been shown that if you have two sets of utilities U_0 and U_0’, then use Bellman eq.s to get U_1 and U_1’, then: $||U_0 - U_0'|| \geq \gamma ||U_1 - U_1'||$

the ∞-norm (i.e. abs max), || [1,-2] – [2, 4] || = 6

Value Iteration Convergence

This means no matter what two sets of utilities you have, they will become “closer” after applying the Bellman update

This is called a contraction and has a nice property you will always converge to a unique solution (when $\gamma<1$)

We can also notice that if U* are the correct utilities, applying Bellman will not change

Value Iteration Convergence

Thus, if U_i is after applying the Bellman eq. i times: $||U_0 - U^*|| \geq \gamma^i ||U_i - U^*||$

But we have a “worst case” utility of:
$U(s_0, s_1, s_2, ...) \leq R_{max}/(1-\gamma)$

Since the difference can at most double this:
$\frac{2 \cdot R_{max}}{1-\gamma} \geq \gamma^i ||U_i - U^*||$

Value Iteration Convergence

If we want to guarantee we are within ε of the optimal solution, we can then find N:

$$\gamma^N \frac{2 \cdot R_{max}}{1-\gamma} \leq \epsilon$$

... as each update contracts/shrinks by γ and we start at most $2*R_{max}/(1-\gamma)$ away from opt.

Also do not need to wait for utility to converge as policy just needs to find best action

Value Iteration Convergence

The Bellman equations find some “utility” for each state that you then find best actions

... but our original goal was:

$$U^{\pi}(s_0, s_1, s_2, ...) = E[\sum_{i=0}^{\infty} \gamma^i R(s_i)]$$

This is similar to the Bellman equations, but Bellman only look one step ahead... while our goal is start to end

Value Iteration Convergence

This is actually not a problem, as if after doing “i” Bellman updates, you have:

$$||U_i - U^*|| < \epsilon$$

... then you are guaranteed (worse case) to be within a a bound of the optimal policy:

$$||U^{\pi_i} - U^*|| < \frac{2 \cdot \epsilon \cdot \gamma}{1 - \gamma}$$

We can the above the “policy loss”

Bellman Equations

Turns out, you can represent state utility in terms of other states (Bellman equation):

$$U(s) = R(s) + \gamma \cdot \max_{a \in Actions} \sum_{s'} P(s'|s,a) \cdot U(s')$$

So for example, the utility of (2,2):

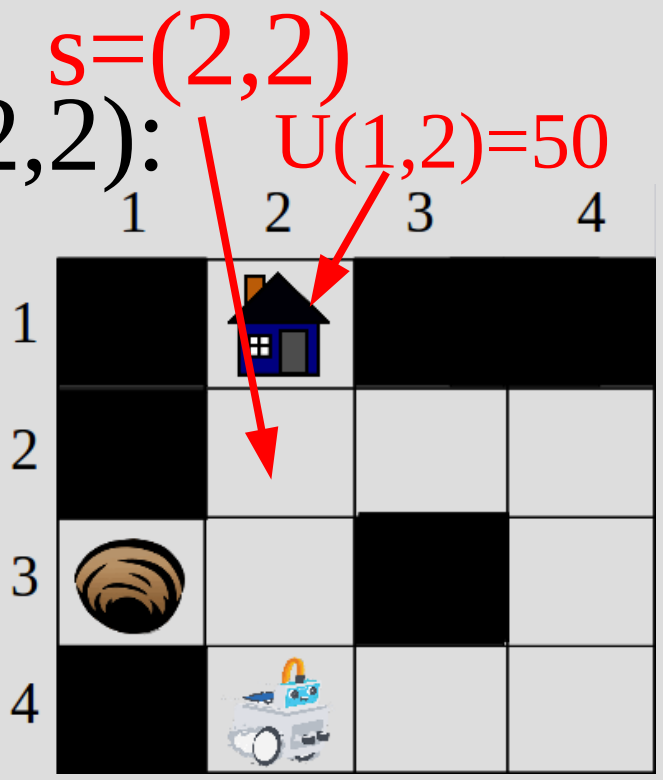
U(2,2) = -1 + γ * max of:

a=Up: $0.8 \cdot 50 + 0.1 \cdot U(2,2) + 0.1 \cdot U(2,3)$

a=D: $0.8 \cdot U(3,2) + 0.1 \cdot U(2,3) + 0.1 \cdot U(2,2)$

a=L: $0.8 \cdot U(2,2) + 0.1 \cdot U(3,2) + 0.1 \cdot 50$

a=R: $0.8 \cdot U(2,3) + 0.1 \cdot 50 + 0.1 \cdot U(3,2)$



Bellman Equations

$$U(s) = R(s) + \gamma \cdot \max_{a \in Actions} \sum_{s'} P(s'|s,a) \cdot U(s')$$

Assuming you should go “up” from (2,2):
(let γ=1.0) ← last time γ=0.9

$$U(2,2) = -1 + 1 \cdot (0.8 \cdot 50 + 0.1 \cdot U(2,2) + 0.1 \cdot U(2,3)$$

Then some algebra:

$$U(2,2) = \frac{39 + 0.1 \cdot U(2,3)}{0.9}$$

... What is U(2,3) assuming best answer is going “left”?

Bellman Equations

$$U(s) = R(s) + \gamma \cdot \max_{a \in Actions} \sum_{s'} P(s'|s,a) \cdot U(s')$$

$$U(2,2) = \frac{39 + 0.1 \cdot U(2,3)}{0.9}$$

U(2,3) going “left” is:

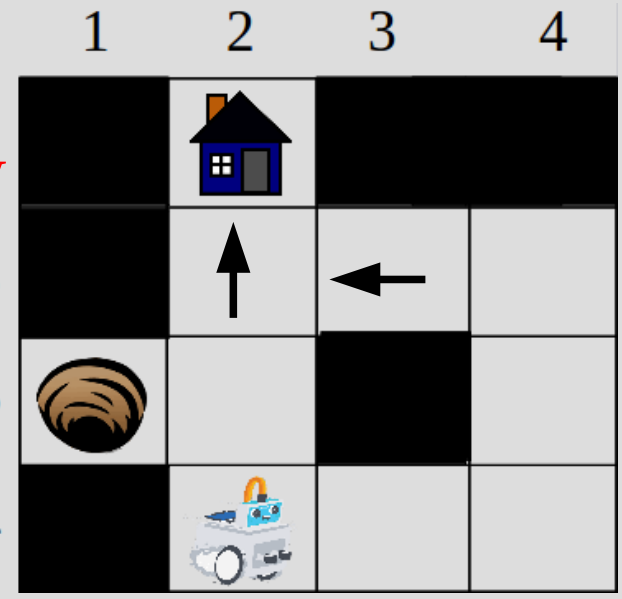
$$U(2,3) = -1 + 1 \cdot (0.8 \cdot U(2,2) + 0.1 \cdot U(2,3) + 0.1 \cdot U(2,3)$$

... algebra ...

$$U(2,3) = \frac{-1 + 0.8 \cdot U(2,2)}{0.8}$$

assumed we knew which actions

Could solve this as sys. linear equations, but we cheated



Policy iteration

This type of problem happens a lot:

- If you knew A, you could solve for B
- If you knew B, you could solve for A
- Yet you know neither A or B

Solution: Initialize A to guess (or random)

1. Solve for B with fixing A
2. Solve for A with fixing B
3. Repeat above 2 until convergence

Policy iteration

So we can actually “cheat” and just assume we know the best direction to move

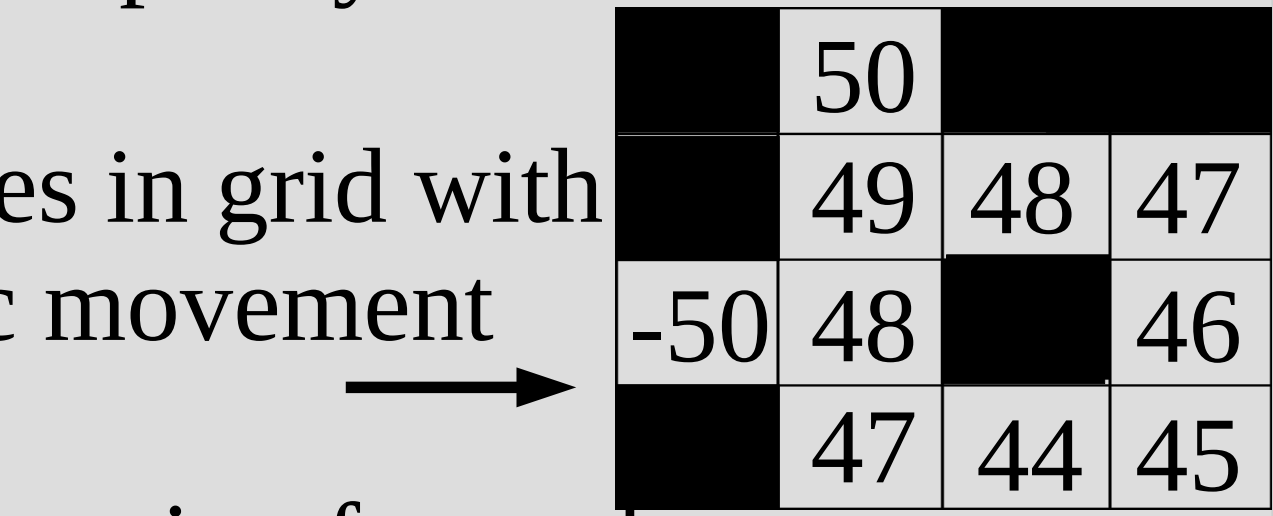
Use this to find the resultant utilities (solving a system of linear equations)

Then once we have utilities, go back and re-find the best directions to move (loop process... though we could have also started with just guessing random utilities)

Policy iteration

We call this method policy iteration

Initialize the values in grid with with deterministic movement



Then we find best action for each square:

$$\underset{a \in \text{actions}}{\text{argmax}} \sum_{s'} P(s'|a,s) \cdot U(s'))$$

only part that depends on action

$$U(s) = R(s) + \gamma \cdot \max_{a \in Actions} \sum_{s'} P(s'|s,a) \cdot U(s')$$

Find best action

$$\underset{a \in \text{actions}}{\operatorname{argmax}} \sum_{s'} P(s'|a,s) \cdot U(s'))$$

	50		
	49	48	47
-50	48		46
	47	44	45

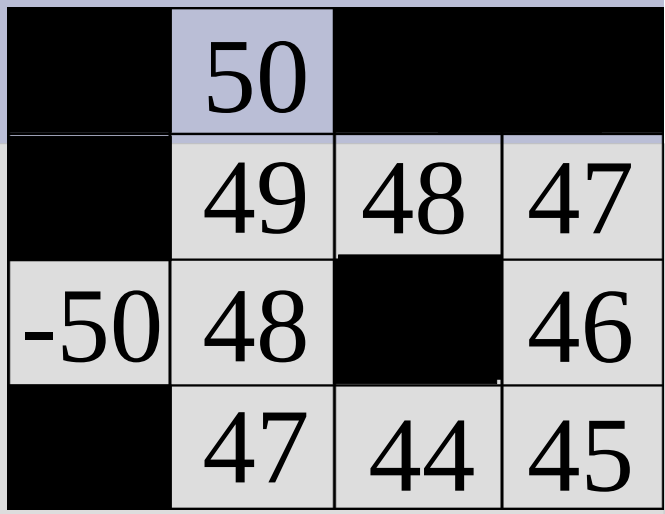
Consider the agent's starting square (the 47)

Find best action (above eq.):

U(2,4) = argmax(Go U, D, L, R)

=argmax((0.8*[U] + 0.1*[L] + 0.1*[R]),
(0.8*[D] + 0.1*[R] + 0.1*[L]),
(0.8*[L] + 0.1*[D] + 0.1*[U]),
(0.8*[R] + 0.1*[U] + 0.1*[D]))

Find best action

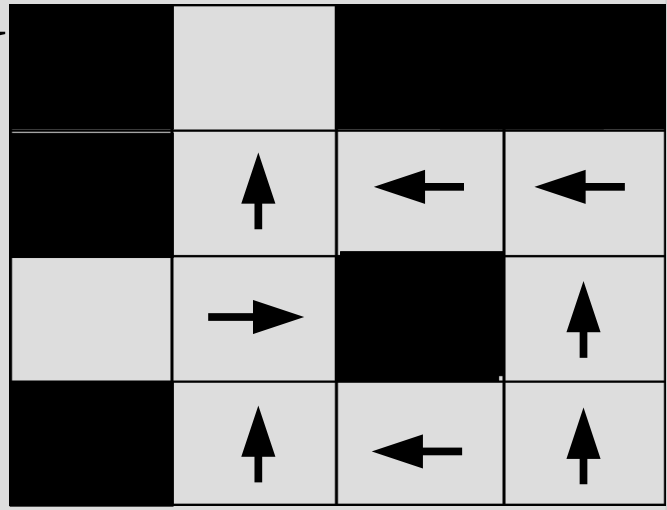


From the 47 (agent start):
[U] = 48, [L] = 47 = [D],
[R] = 44,

argmax((0.8*48 + 0.1*47 + 0.1*44),
(0.8*47 + 0.1*44 + 0.1*47),
(0.8*47 + 0.1*47 + 0.1*48),
(0.8*44 + 0.1*48 + 0.1*47))
=argmax(47.5, 46.7, 47.1, 44.7)
=Go U

Find values

We repeat this process for every square and get a “best action” grid



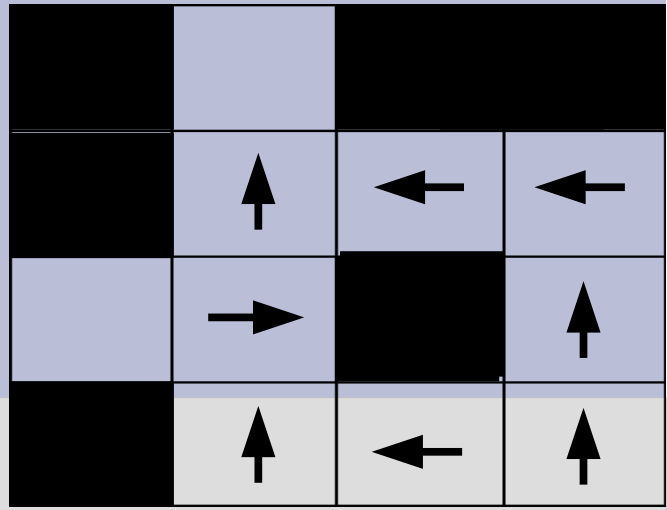
Then use the Bellman eq. with fixed actions to get system of linear equations (each state is 1 unknown value with 1 equation)

$$U(s) = R(s) + \gamma \cdot \max_{a \in Actions} \sum_{s'} P(s'|s,a) \cdot U(s')$$

assuming action removes max... makes linear

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

Find values



U(1,2)= +50 (goal)

U(2,2)=-1+0.8*U(1,2)+0.1*U(2,2)+0.1*U(2,3)

U(2,3)=-1+0.8*U(2,2)+0.1*U(2,3)+0.1*U(2,3)

U(2,4)=-1+0.8*U(2,3)+0.1*U(3,4)+0.1*U(2,4)

U(3,1)= -50 (pit)

U(3,2)=-1+0.8*U(3,2)+0.1*U(2,2)+0.1*U(4,2)

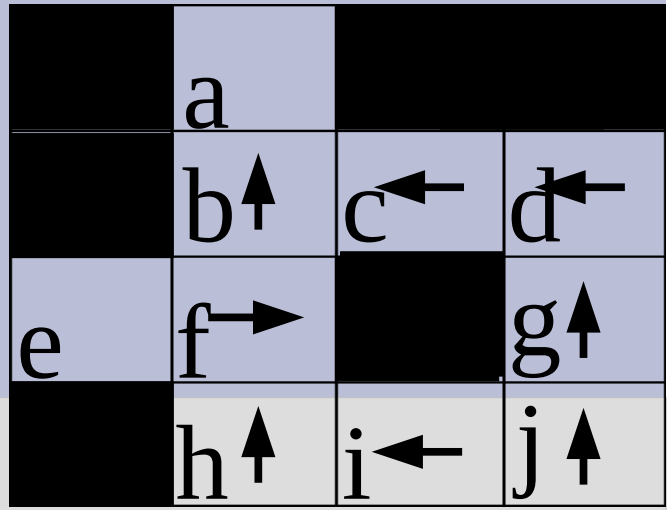
U(3,4)=-1+0.8*U(2,4)+0.1*U(3,4)+0.1*U(3,4)

U(4,2)=-1+0.8*U(3,2)+0.1*U(4,2)+0.1*U(4,3)

U(4,3)=-1+0.8*U(4,2)+0.1*U(4,3)+0.1*U(4,3)

U(4,4)=-1+0.8*U(3,4)+0.1*U(4,3)+0.1*U(4,4)

Find values



a = 50

b=-1+0.8*a+0.1*b+0.1*c

c=-1+0.8*b+0.1*c+0.1*c

d=-1+0.8*c+0.1*g+0.1*d

e= -50

f=-1+0.8*f+0.1*b+0.1*h

g=-1+0.8*d+0.1*g+0.1*g

h=-1+0.8*f+0.1*h+0.1*i

i=-1+0.8*h+0.1*i+0.1*i

j=-1+0.8*g+0.1*i+0.1*j

Find values

Solving that mess gives you these new values:

	50		
	48.59	47.34	45.93
-50	37.18		44.68
	35.78	34.53	42.44

At this point, you would again find the best move for the values above and repeat until the actions do not change

Policy Iteration

Now you do it!

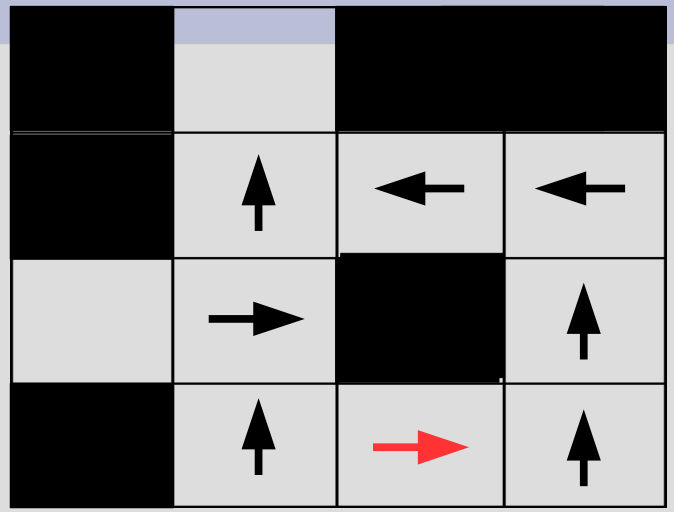
$$U(s) = R(s) + \gamma \cdot \sum_{s'} P(s'|s,a) \cdot U(s')$$

1. Find the best actions for these values
2. If any actions changed, setup sys. lin. eq.
   (otherwise you know best paths)

	50		
	48.59	47.34	45.93
-50	37.18		44.68
	35.78	34.53	42.44

Policy Iteration

1.



2.

	50		
	48.59	47.34	45.93
-50	37.93		44.68
	37.28	42.03	43.28

Policy Iteration

After 1 more system of linear equations, the actions stabilize and we find that we should go around the long way to the goal

(i.e. pit is too dangerous)

The starting node will have a value of 40.6526, so it will take approximately 9.34743 steps to reach the goal (using optimal actions at each step)