

18-0: **Making Sequential Decisions**

- Previously, we've talked about:
  - Making one-shot decisions in a deterministic environment
  - Making sequential decisions in a deterministic environment
    - Search
    - Inference
  - Making one-shot decisions in a stochastic environment
    - Probability and Belief Networks
    - Expected Utility
- What about sequential decisions in a stochastic environment?

18-1: **Expected Utility**

- Recall that the expected utility of an action is the utility of each possible outcome, weighted by the probability of that outcome occurring.
- More formally, from state  $s$ , an agent may take actions  $a_1, a_2, \dots, a_n$ .
- Each action  $a_i$  can lead to states  $s_{i1}, s_{i2}, \dots, s_{im}$ , with probability  $p_{i1}, p_{i2}, \dots, p_{im}$

$$EU(a_i) = \sum p_{ij} s_{ij}$$

- We call the set of probabilities and associated states the state transition model.
- The agent should choose the action  $a'$  that maximizes EU.

18-2: **Markovian environments**

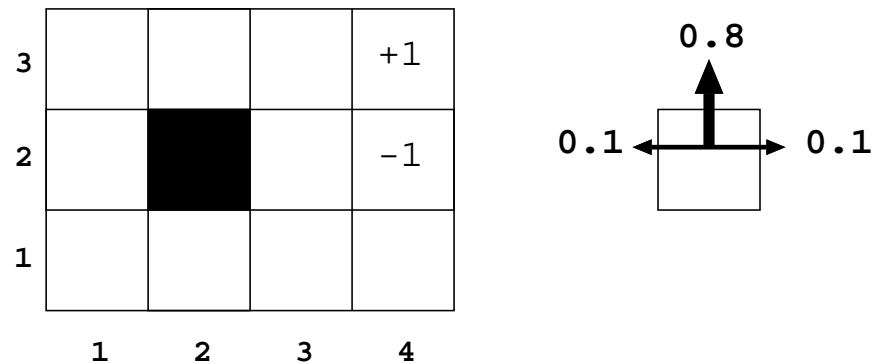
- We can extend this idea to sequential environments.
- Problem: How to determine transition probabilities?
  - The probability of reaching state  $s$  given action  $a$  might depend on previous actions that were taken.
  - Reasoning about long chains of probabilities can be complex and expensive.
- The Markov assumption says that state transition probabilities depend only on a finite number of parents.
- Simplest: a first-order Markov process. State transition probabilities depend only on the previous state.
  - This is what we'll focus on.

18-3: **Stationary Distributions**

- We'll also assume a *stationary distribution*
- This says that the probability of reaching a state  $s'$  given action  $a$  from state  $s$  with history  $H$  does not change.
- Different histories may produce different probabilities
- Given identical histories, the state transitions will be the same.
- We'll also assume that the utility of a state does not change throughout the course of the problem.
  - In other words, our model of the world does not change while we are solving the problem.

18-4: **Solving sequential problems**

- If we have to solve a sequential problem, the total utility will depend on a sequence of states  $s_1, s_2, \dots, s_n$ .
- Let's assign each state a utility or *reward*  $R(s_i)$ .
- Agent wants to maximize the sum of rewards.
- We call this formulation a Markov decision process.
  - Formally:
  - An initial state  $s_0$
  - A discrete set of states and actions
  - A Transition model:  $T(s, a, s')$  that indicates the probability of reaching state  $s'$  from  $s$  when taking action  $a$ .
  - A reward function:  $R(s)$



#### 18-5: Example grid problem

- Agent moves in the “intended” direction with probability 0.8, and at a right angle with probability 0.2
- What should an agent do at each state to maximize reward?

#### 18-6: MDP solutions

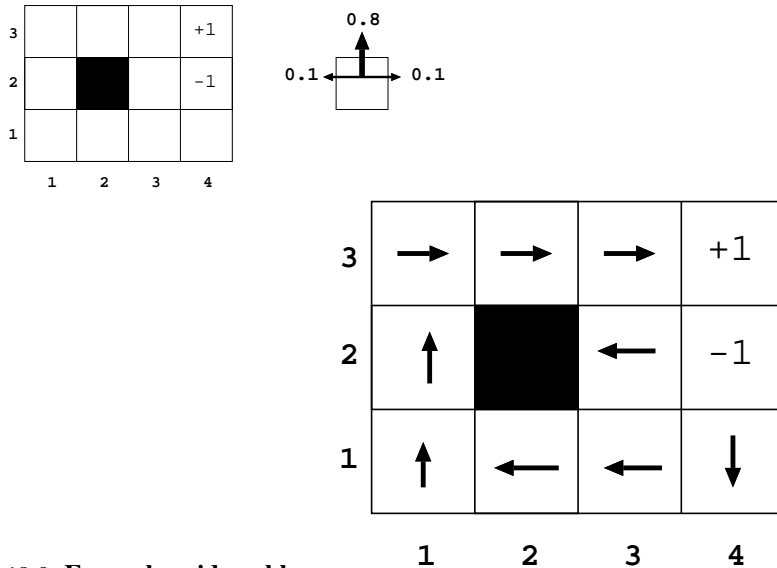
- Since the environment is stochastic, a solution will not be an action sequence.
- Instead, we must specify what an agent should do in any reachable state.
- We call this specification a *policy*
  - “If you’re below the goal, move up.”
  - “If you’re in the left-most column, move right.”
- We denote a policy with  $\pi$ , and  $\pi(s)$  indicates the policy for state  $s$ .

#### 18-7: MDP solutions

- Things to note:
  - We’ve wrapped the goal formulation into the problem
    - Different goals will require different policies.
  - We are assuming a great deal of (correct) knowledge about the world.
    - State transition models, rewards
    - We’ll touch on how to learn these without a model.

18-8: Comparing policies

- We can compare policies according to the expected utility of the histories they produce.
- The policy with the highest expected utility is the *optimal policy*.
- Once an optimal policy is found, the agent can just look up the best action for any state.



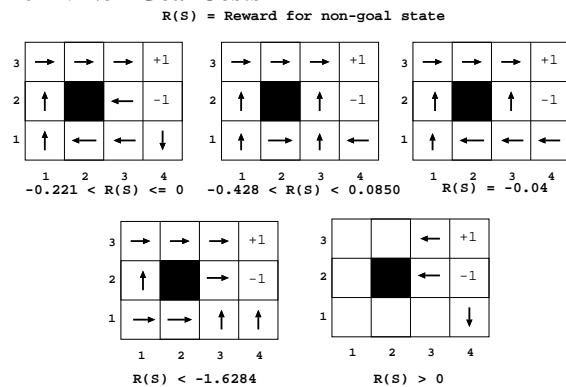
18-9: Example grid problem

- Assumes no cost for non-goal states
- No benefit for faster solutions

18-10: Non-Goal Costs

- Spending unlimited time trying to find th best solution is not always the best idea.
- We can give a cost (negative utility) to each non-goal state
- penalized for taking too long to find the goal state

18-11: Non-Goal Costs



- Very high cost: Agent tries to exit immediately

- Middle ground: Agent tries to avoid bad exit
- Positive reward: Agent doesn't try to exit.

#### 18-12: More on reward functions

- In solving an MDP, an agent must consider the value of future actions.
- There are different types of problems to consider:
- Horizon - does the world go on forever?
  - Finite horizon: after  $N$  actions, the world stops and no more reward can be earned.
  - Infinite horizon; World goes on indefinitely, or we don't know when it stops.
    - Infinite horizon is simpler to deal with, as policies don't change over time.

#### 18-13: More on reward functions

- We also need to think about how to value future reward.
- \$100 is worth more to me today than in a year.
- We model this by *discounting* future rewards.
  - $\gamma$  is a *discount factor*
- $U(s_0, s_1, s_2, s_3, \dots) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 R(s_3) + \dots, \gamma \in [0, 1]$
- If  $\gamma$  is large, we value future states
- if  $\gamma$  is low, we focus on near-term reward
- In monetary terms, a discount factor of  $\gamma$  is equivalent to an interest rate of  $(1/\gamma) - 1$

#### 18-14: More on reward functions

- Discounting lets us deal sensibly with infinite horizon problems.
  - Otherwise, all EUs would approach infinity.
- Expected utilities will be finite if rewards are finite and bounded and  $\gamma < 1$ .
- We can now describe the optimal policy  $\pi^*$  as:

$$\pi^* = \operatorname{argmax}_{\pi} EU\left(\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi\right)$$

#### 18-15: Value iteration

- How to find an optimal policy?
- We'll begin by calculating the expected utility of each state and then selecting actions that maximize expected utility.
- In a sequential problem, the utility of a state is the expected utility of all the state sequences that follow from it.
- This depends on the policy  $\pi$  being executed.

- Essentially,  $U(s)$  is the expected utility of executing an optimal policy from state  $s$ .

<b>3</b>	0.812	0.868	0.918	<b>+1</b>
<b>2</b>	0.762		0.660	<b>-1</b>
<b>1</b>	0.705	0.655	0.611	0.388
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>

$\gamma = 1$   
 $R(s) = -0.04$   
 (non-goals)

18-16: **Utilities of States**

- Notice that utilities are highest for states close to the +1 exit.

18-17: **Utilities of States**

- The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.

- 

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

- This is called the Bellman equation

- Example:

$$\begin{aligned}
 U(1, 1) = & -0.04 + \gamma \max(0.8U(1, 2) + 0.1U(2, 1) + 0.1U(1, 1) \\
 & 0.9U(1, 1) + 0.1U(1, 2), \\
 & 0.9U(1, 1) + 0.1U(2, 1), \\
 & 0.8U(2, 1) + 0.1U(1, 2) + 0.1U(1, 1))
 \end{aligned}$$

18-18: **Dynamic Programming**

- Solving the Bellman equation is a dynamic programming problem
- In an acyclic transition graph, you can solve these recursively by working backward from the final state to the initial states.
- Can't do this directly for transition graphs with loops.

18-19: **Value Iteration**

- Since state utilities are defined in terms of other state utilities, how to find a closed-form solution?
- We can use an iterative approach:
  - Give each state random initial utilities.
  - Calculate the new left-hand side for a state based on its neighbors' values.
  - Propagate this to update the right-hand-side for other states,

- Update rule:  $U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s')$

- This is guaranteed to converge to the solutions to the Bellman equations.

18-20: Value Iteration algorithm

Assigning random utilities to each state

do

  for s in states

$U(s) = R(s) + \gamma \max T(s, a, s') U(s')$

until

  all utilities change by less than delta

- where  $\delta = error * (1 - \gamma) / \gamma$

3	0.1	-0.1	0.05	+1
2	-0.02		0.15	-1
1	0.0	0.1	-0.1	0.15
	1	2	3	4

$\gamma = 0.8$   
 $R(s) = -0.04$   
 error = 0.01  
 $\delta = 0.0025$

18-21: Example

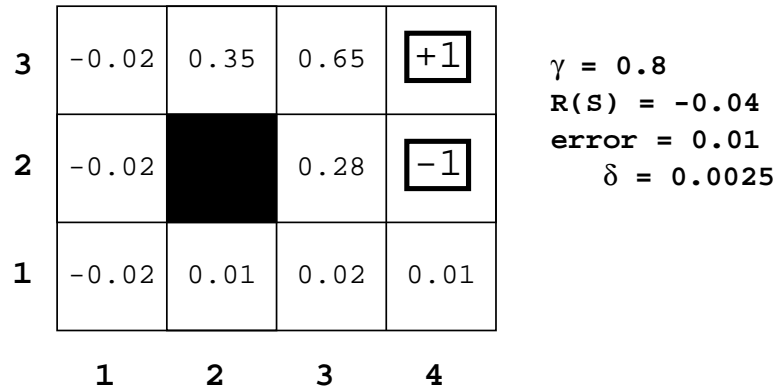
- Initially, use random values

3	0.03	-0.02	0.62	+1
2	0.02		0.05	-1
1	0.02	0.02	0.08	0.06
	1	2	3	4

$\gamma = 0.8$   
 $R(s) = -0.04$   
 error = 0.01  
 $\delta = 0.0025$

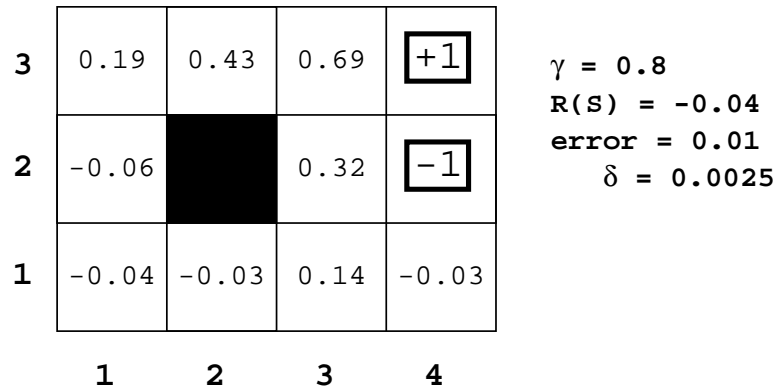
18-22: Example

- After 1 iteration



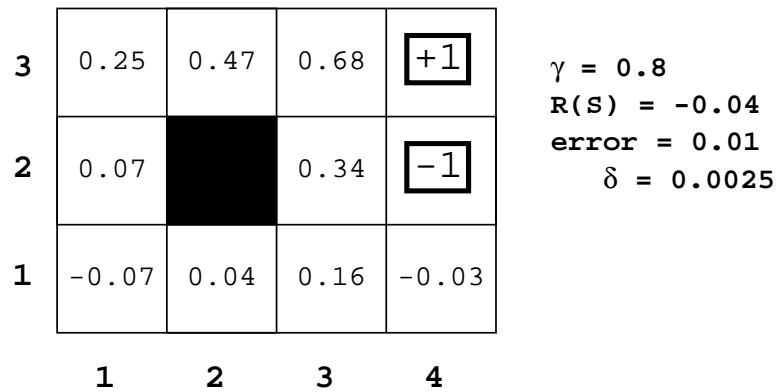
18-23: Example

- After 2 iterations



18-24: Example

- After 3 iterations



18-25: Example

- After 4 iterations

3	0.27	0.47	0.68	$\boxed{+1}$	$\gamma = 0.8$ $R(S) = -0.04$ $\text{error} = 0.01$ $\delta = 0.0025$
2	0.13		0.34	$\boxed{-1}$	
1	0.0	0.07	0.18	-0.02	
	1	2	3	4	

18-26: **Example**

- After 5 iterations

3	0.29	0.47	0.68	$\boxed{+1}$	$\gamma = 0.8$ $R(S) = -0.04$ $\text{error} = 0.01$ $\delta = 0.0025$
2	0.15		0.34	$\boxed{-1}$	
1	0.04	0.08	0.18	-0.01	
	1	2	3	4	

18-27: **Example**

- After 6 iterations – almost converged

18-28: **Discussion**

- Strengths of Value iteration
  - Guaranteed to converge to correct solution
  - Simple iterative algorithm
- Weaknesses:
  - Convergence can be slow
  - We really don't need all this information
  - Just need *what to do* at each state.

18-29: **Policy iteration**

- Policy iteration helps address these weaknesses.
- Searches directly for optimal policies, rather than state utilities.
- Same idea: iteratively update policies for each state.
- Two steps:
  - Given a policy, compute the utilities for each state.



- Compute a new policy based on these new utilities.

18-30: **Policy iteration algorithm**

```

Initialize all state utilities to zero
Pi = random policy vector indexed by state
do
  U = evaluate the utility of each state for Pi
  for s in states
    a = find action that maximizes expected
        utility for that state
    Pi(s) = a
while some action changed
    
```

3	0.0 ↓	0.0 →	0.0 ↓	<b>+1</b>
2	0.0 ←		0.0 →	<b>-1</b>
1	0.0 →	0.0 ←	0.0 ↑	0.0 ←
	1	2	3	4

**All non-goal utilities 0**  
**Random policies**

18-31: **Policy Iter. Example**

3	-0.04 ↓	-0.04 →	0.04 ↓	<b>+1</b>
2	-0.04 ←		-0.68 →	<b>-1</b>
1	-0.04 →	-0.04 ←	-0.04 ↑	-0.12 ←
	1	2	3	4

**Assign new utilities based on old utilities and policy**

18-32: **Policy Iter. Example**

3	-0.07 ↓	-0.07 →	0.04 →	<b>+1</b>
2	-0.07 ←		-0.68 ↑	<b>-1</b>
1	-0.07 →	-0.07 ←	-0.04 ←	-0.12 ←
	1	2	3	4

**Create a new policy based on new Utilities**

18-33: Policy Iter. Example

3	-0.7 ↓	-0.02 →	0.55 →	<b>+1</b>
2	-0.07 ←		-0.14 ↑	<b>-1</b>
1	-0.07 →	-0.07 ←	-0.12 ←	-0.12 ←
	1	2	3	4

**Create new Utilities based on policy and previous Utilities**

18-34: Policy Iter. Example

3	-0.07 →	-0.02 →	0.55 →	<b>+1</b>
2	-0.07 ←		-0.14 ↑	<b>-1</b>
1	-0.07 →	-0.07 ←	-0.12 ←	-0.12 ↓
	1	2	3	4

**Create policies based on previous Utilities**

18-35: Policy Iter. Example

3	-0.06 →	0.31 →	0.63 →	<b>+1</b>
2	-0.09 ←		0.22 ↑	<b>-1</b>
1	-0.09 →	-0.09 ←	-0.10 ←	-0.12 ↓
	1	2	3	4

**Create new Utilities using old Utilities and Policy**

18-36: Policy Iter. Example

3	-0.06 →	0.31 →	0.63 →	<b>+1</b>
2	-0.09 ↑		0.22 ↑	<b>-1</b>
1	-0.09 →	-0.09 ←	-0.10 ↑	-0.12 ↓
	1	2	3	4

**Use new utility estimates to construct new policies**

18-37: Policy Iter. Example

3	0.15 →	0.41 →	0.67 →	<b>+1</b>
2	0.04 ↑		0.30 ↑	<b>-1</b>
1	-0.11 →	-0.11 ←	0.08 ↑	-0.13 ↓
	1	2	3	4

**Create new utility estimates using old Utilities & current policies**

18-38: Policy Iter. Example

3	0.15 →	0.41 →	0.67 →	<b>+1</b>
2	0.04 ↑		0.30 ↑	<b>-1</b>
1	-0.11 ↑	-0.11 →	0.08 ↑	-0.13 ←
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>

**Use new Utilities to update policy**

18-39: Policy Iter. Example

3	0.23 →	0.45 →	0.68 →	<b>+1</b>
2	0.06 ↑		0.33 ↑	<b>-1</b>
1	-0.03 ↑	-0.01 →	0.13 ↑	-0.07 ←
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>

**Update utilities based on old Utilities and Policies**

18-40: Policy Iter. Example

3	0.23 →	0.45 →	0.68 →	<b>+1</b>
2	0.06 ↑		0.33 ↑	<b>-1</b>
1	-0.03 ↑	-0.01 →	0.13 ↑	-0.07 ←
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>

**Update policies  
No change.**

18-41: Policy Iter. Example

18-42: Discussion

- Advantages:
  - Faster convergence.
  - Solves the actual problem we're interested in. We don't really care about utility estimates except as a way to construct a policy.

18-43: **Learning a Policy**

- MDPs assume that we know a model of the world
  - Specifically, the transition function  $T$
- We can also learn a policy through interaction with the environment.
- This is known as *reinforcement learning*.
- We'll talk about this in a couple of weeks.

18-44: **Summary**

- Markov decision policies provide an agent with a description of *how to act optimally* for any state in a problem.
  - Must know state space, have a fixed goal.
- Value iteration and policy iteration can be applied to solve this.