Tufts

Class #21: Markov Decision Processes as Models for Learning

Machine Learning (COMP 135): M. Allen, 18 Nov. 19

1

What Do We Want AI and ML to Do?

- Short answer: Lots of things!
  - Intelligent robot and vehicle navigation
  - Better web search
  - Automated personal assistants
  - Scheduling for delivery vehicles, air traffic control, industrial processes, …
  - Simulated agents in video games
  - Automated translation systems

2

What Do We Need?

- AI systems must be able to handle complex, uncertain worlds, and come up with plans that are useful to us over extended periods of time
  - ***Uncertainty***: requires something like probability theory
  - ***Value-based planning***: we want to maximize expected utility over time, as in decision theory
  - ***Planning over time***: we need some sort of temporal model of how the world can change as we go about our business

3

Markov Decision Processes

- Markov Decision Processes (MDPs) combine various ideas from probability theory and decision theory
  - A useful model for doing full ***planning***, and for representing environments where agents can ***learn*** what to do
- Basic idea: a world made up of states, changing based on the actions of an AI agent, who is trying to maximize its long-term reward as it does so
  - One technical detail: change happens ***probabilistically*** (under the Markov assumption)

4

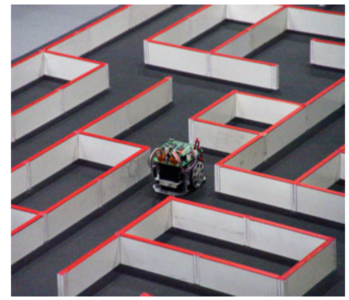
Formal Definition of an MDP

- An MDP has several components

$$M = < S,\ A,\ P,\ R,\ T >$$

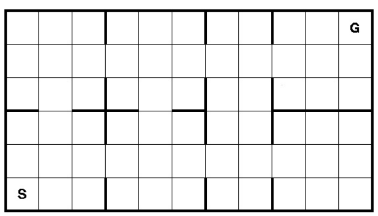
1. S = a set of ***states*** of the world
2. A = a set of ***actions*** an agent can take
3. P = a ***state-transition function***: $P\,(s,\ a,\ s')$ is the probability of ending up in state s' if you start in state s and you take action a: $\mathrm{P}(s' \mid s,\ a\,)$
4. R = a ***reward function***: $R(s,\ a,\ s')$ is the one-step reward you get if you go from state s to state s' after taking action a
5. T = a ***time horizon*** (how many steps): we assume that every state-transition, following a single action, takes a single unit of time

An Example: Maze Navigation



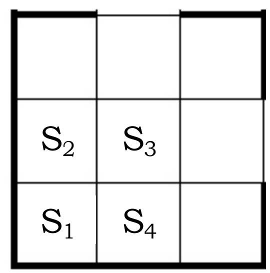
- Suppose we have a robot in a maze, looking for exit
- The robot can see where it is currently, and where surrounding walls are, but doesn't know anything else
- We would like it to be able to learn the shortest route out of the maze, no matter where it starts
- How can we formulate this problem as an MDP?

MDP for the Maze Problem



- ***States***: each state is simply the robot's current location (imagine the map is a grid), including nearby walls
- ***Actions***: the robot can move in one of the four directions (UP, DOWN, LEFT, RIGHT)

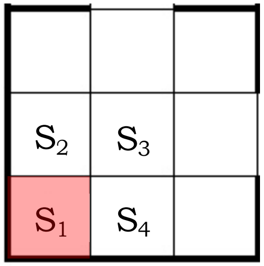
Action Transitions



- We can use the transition function to represent important features of the maze problem domain
- For instance, the robot cannot move through walls
- For example, if the robot starts in the corner (s_1), and tries to go DOWN, nothing happens:

$$\mathrm{P}(s_1,\ \mathrm{DOWN},\ s_1) = 1.0$$

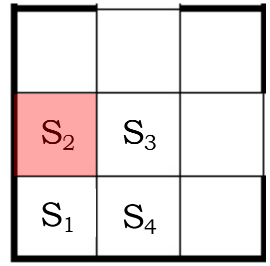
Action Transitions, II



- Similarly, we can model ***uncertain action outcomes*** using the transition model
- Suppose the robot is a little unstable, and occasionally goes in the wrong direction
- Thus, if it starts in state s_1 and tries to go UP to s_2:

9

Action Transitions, II

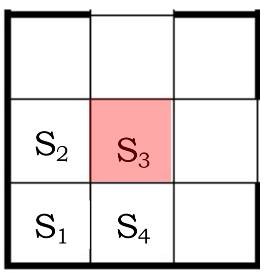


- Similarly, we can model ***uncertain action outcomes*** using the transition model
- Suppose the robot is a little unstable, and occasionally goes in the wrong direction
- Thus, if it starts in state s_1 and tries to go UP to s_2:
  1. 80% of the time ***it works***:

     $P(s_1, \text{UP}, s_2) = 0.8$

10

Action Transitions, II



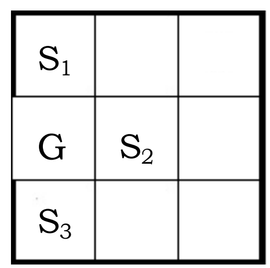
- Similarly, we can model ***uncertain action outcomes*** using the transition model
- Suppose the robot is a little unstable, and occasionally goes in the wrong direction
- Thus, if it starts in state s_1 and tries to go UP to s_2:
  1. 80% of the time it works:

     $P(s_1, \text{UP}, s_2) = 0.8$
  2. But it may ***slip and miss***:

     $P(s_1, \text{UP}, s_3) = 0.2$

11

Rewards in the Maze



- If G is our **goal (exit) state**, we can "encourage" the robot, by giving any action that gets to G positive reward:

  $R(s_1, \text{DOWN}, G) = +100$

  $R(s_2, \text{LEFT}, G) = +100$

  $R(s_3, \text{UP}, G) = +100$
- Further, we can reward ***quicker solutions*** by making all other movements have negative reward, e.g.:

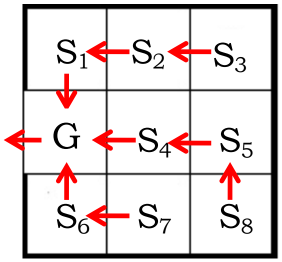
  $R(s_1, \text{RIGHT}, s') = -1$

  $R(s_2, \text{UP}, s') = -1$

  etc.

12

Solving the Maze



- A ***solution*** to our problem takes the form of a policy of action, π
- At each state, it tells the agent the best thing to do:
  - $\pi(s_1)$ = DOWN
  - $\pi(s_2)$ = LEFT
  - Similarly for all other states…

Planning and Learning

- How do we ***find*** policies?
- If we ***know the entire problem***, we plan
  - e.g., if we ***already know*** the whole maze, and know all the MDP dynamics, we can solve it to find the best policy of action (even if we have to take into account the probability that some movements fail some of the time)
- If we ***don't know it all*** ahead of time, we learn
  - Reinforcement Learning: use the positive and negative feedback from the ***one-step*** reward in an MDP, and figure out a policy that gives us ***long-term*** value

Maximizing Expected Return

- If we are solving a planning problem like an MDP, we want our plan to give us maximum expected reward over time
- In a finite-time problem, the total reward we get at some time-step t is just the sum of future rewards (up to our time-limit T):

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

- The optimal policy would make this sum as large as possible, taking into account any probabilistic outcomes (e.g. robot moves that go the wrong way by accident)

The Infinite (Indefinite) Case

- Unfortunately, this simple idea doesn't really work for problems with indefinite time-horizons
- In such problems, our agent can keep on acting, and we have ***no known upper bound*** on how long this may continue
  - In such cases we treat upper bound as if it is ***infinite***: $T = \infty$
- If the time-horizon T is infinite, then the sum of rewards:

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

can be infinitely large (or infinitely small), too!

The Infinite (Indefinite) Case

- If the time-horizon T is infinite, then the sum of rewards:

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

can be infinitely large (or infinitely small), too!

- For example, suppose a robot is exploring Mars
  - Whenever it collects a valuable sample, it gets a reward of $+100$
  - Less valuable samples only give it $+1$ (everything else is just 0)
- Now, if the problem is indefinite-horizon, it ***doesn't matter*** what the robot does: ***all policies*** give it the same value ($+\infty$) even if it ***ignores*** any valuable samples

Discounted Reward

- To solve the problem of future reward in MDPs, we therefore introduce a discount rate, γ (gamma), which is some number between 0 and 1
- Reward we get is then ***weighted*** by the discount rate:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- If our time horizon is finite, we can set gamma to 1; if it is infinite, we always make sure that gamma is ***less than*** 1
- What happens if gamma = 0?

Policy Values in MDPs

- Suppose we have a policy π for an MDP
- A policy is a ***function from states to actions***

$$\pi : S \rightarrow A$$

- We can calculate the expected value we get by starting in some state s, at time t, and then following policy π for T steps:

$$U^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\left\{\sum_{k=0}^{T-1} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

  - $E_\pi\{\ldots\}$ is the expectation for value of $\{\ldots\}$ if we follow policy π
  - If the domain is not probabilistic, then we know this exactly, otherwise we can calculate it using probability/decision theory

The Bellman Equation

- Using basic algebra, the expected value of starting in some state, s, can be calculated, via dynamic programming, based on the ***next*** possible state(s) we can reach if we take the action dictated by our policy, $\pi(s) = a$:

$$\begin{aligned}
U^\pi(s) &= E_\pi\{R_t \mid s_t = s\} \\
&= E_\pi\left\{\sum_{k=0}^{T-1} \gamma^k r_{t+k+1} \middle| s_t = s\right\} \\
&= E_\pi\left\{r_{t+1} + \gamma \sum_{k=0}^{T-2} \gamma^k r_{t+k+2} \middle| s_t = s\right\} \\
&= \sum_{s'} P(s,\pi(s),s') \left[R(s,\pi(s),s') + \gamma\, E_\pi\left\{\sum_{k=0}^{T-2} \gamma^k r_{t+k+2} \middle| s_{t+1} = s'\right\}\right]
\end{aligned}$$

- Which means that we can define policy-value for state s ***recursively***, based on the policy-value of any next state s' that we can get to when we follow that policy:

$$U^\pi(s) = \sum_{s'} P(s,\pi(s),s')\,[R(s,\pi(s),s') + \gamma\, U^\pi(s')]$$

Unpacking the Bellman Equation

- Derived first by Richard Bellman (1957), working in control theory
  - He also showed how to calculate the value of the equation
- Defines policy-value for state s ***recursively***, based on the policy-value of any next state s' that we can get to when we follow that policy:

$$U^\pi(s) = \sum_{s'} P(s, \pi(s), s') \, [R(s, \pi(s), s') + \gamma \, U^\pi(s')]$$

Expected value of following policy π, starting in state s

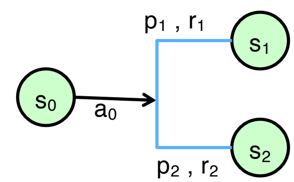
Sum over all possible next states, s'

Transition probability of going from s to s', following action $\pi(s)$

One-step reward for going from s to s', following action $\pi(s)$

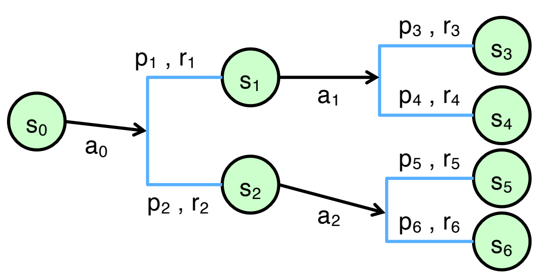
Discounted value of ***continuing*** to follow policy π, from state s'

Bellman Updates



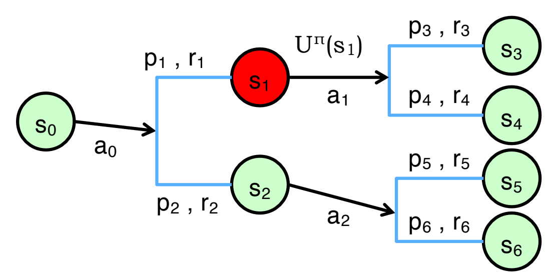
- Consider a 2-step policy, π, starting in state s_0
- At step 1, we take action $a_0 = \pi(s_0)$, which leads to some possible next states, s_1 or s_2, each with different probabilities and resulting rewards

Bellman Updates



- Then, each of these next states also has some action to take under our policy, leading to further transitions and rewards gained over time

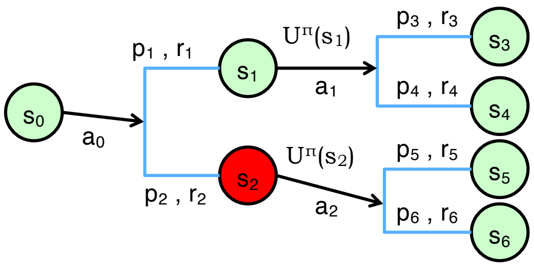
Bellman Updates



- Thus, to get the value of the start-state under this policy, $U^\pi(s_0)$, we first calculate one-step, undiscounted expected value:

$$U^\pi(s_1) = (p_3 \times r_3) + (p_4 \times r_4)$$

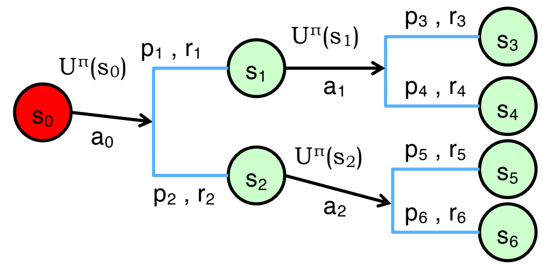
Bellman Updates



- Similarly, we have:

$$U^{\pi}(s_2) = (p_5 \times r_5) + (p_6 \times r_6)$$

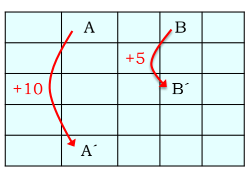
Bellman Updates

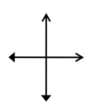


- Now we can calculate our start-state value, but this time discounting the value of the next states by our γ factor:

$$U^{\pi}(s_0) = (p_1 \times [r_1 + \gamma U^{\pi}(s_1)]) + (p_2 \times [r_2 + \gamma U^{\pi}(s_2)])$$

Solving the Bellman Equation





Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

- Next, we will see how to ***solve*** the general Bellman Equation for any set of states, probabilities, and rewards, over any time horizon
- Here, we see the solution for a grid with dynamics as follows:
  - Agent policy: ***move randomly*** in one of 4 directions
  - If agent hits a wall, reward is $R = -1$
  - All other moves are reward $R = 0$, except for in two special states A and B, where any action takes agent to A´ or B´ with reward indicated
  - Discount factor (gamma) is $\gamma = 0.9$

Example from: Sutton & Barto, 1998

Evaluating a Policy Iteratively

function POLICY-EVALUATION(mdp, π) **returns** a value function
  inputs: mdp, an MDP, and π, a policy to be evaluated
  local variables: Δ, maximal amount policy values change per iteration,
    Θ, a small positive constant

  $\forall s \in S: \; U(S) = 0$
  repeat while $\Delta \geq \Theta$
    $\Delta \leftarrow 0$
    $\forall s \in S:$
      $u \leftarrow U(s)$
      $U(s) \leftarrow \sum_{s'} P(s, \pi(s), s')[R(s, \pi(s), s') + \gamma\, U(s')]$
      $\Delta \leftarrow \max(\Delta, |U(s) - u|)$

  return value function $U \approx U^{\pi}$

- Policy evaluation: given a policy, we calculate the expected value for every state if we follow the policy, iterating until values converge (quit changing much)

Next Few Weeks

- **Topics**: Reinforcement Learning
- **HW 05**: due Wednesday, 20 November, 9:00 AM
- **Project 02**: due Monday, 25 November, 9:00 AM
- **Office Hours**: 237 Halligan, Tuesday, 11:00 AM – 1:00 PM
  - TA hours can be found on class website as well