Outline

• Sequential decision problems.
• Value iteration
• Policy iteration
Sequential decision problems

Planning

Decision-theoretic planning

uncertainty and utility

explicit actions and subgoals

Search

Markov decision problems (MDPs)

uncertainty and utility

explicit actions and subgoals

Partially observable MDPs (POMDPs)

uncertain sensing

(belief states)
Markov Decision Processes (MDP)

- A fundamental framework in Artificial Intelligence (AI).

- History
  - 1950s: early works of Bellman and Howard
  - 50s-80s: theory, basic set of algorithms, applications
  - 90s: MDPs in AI literature

- MDPs in AI
  - Probabilistic Planning (today!)
  - Reinforcement Learning (next Lecture)
Example MDP

States $s \in S$, actions $a \in A$ such that $A = \{Up, Down, Left, Right\}$

Model $T(s, a, s') \equiv P(s'|s, a) =$ probability that $a$ in $s$ leads to $s'$

Reward function $R(s)$ (or $R(s, a), R(s, a, s')$)

\[
= \begin{cases} 
-0.04 & \text{(small penalty) for nonterminal states} \\
\pm 1 & \text{for terminal states}
\end{cases}
\]
Solving MDPs

- In search problems, aim is to find an optimal sequence.
- In MDPs, aim is to find an optimal policy $\pi(s)$ i.e., best action for every possible state $s$ (because can't predict where one will end up).
- The optimal policy maximizes (say) the expected sum of rewards.
- Optimal policy when state penalty $R(s)$ is -0.04:
Risk and reward

The balance between risk and reward changes depending of the value of $R(s)$ for nonterminal states.

$r = [-\infty : -1.6284]$  \hspace{1cm}  $r = [-0.4278 : -0.0850]$  

$r = [-0.0480 : -0.0274]$  \hspace{1cm}  $r = [-0.0218 : 0.0000]$
Risk and reward

The careful balance between risk and reward is a characteristic of MDPs that does not arise in deterministic search problems. This is a characteristic of many real-world problems.
Utility of state sequences

- Need to understand preferences between sequences of states.
- Typically consider *stationary preferences* on reward sequences:

\[ [r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots] \equiv [r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots] \]

**Observation:** There are only two ways to combine rewards over time.

1) *Additive* utility function:

\[ U([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots \]

2) *Discounted* utility function:

\[ U([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \]

where \( \gamma \) is the discount factor.
Utility of states

Utility of a state (a.k.a. its value) is defined to be

\[ U(s) = \text{expected (discounted) sum of rewards (until termination) assuming optimal actions} \]

Given the utilities of the states, choosing the best action is just Maximize the Expected Utility (MEU) of the immediate successors.
Utilities contd.

Problem: infinite lifetimes ⇒ additive utilities are infinite

1. **Finite horizon**: termination at a fixed time $T$ ⇒ nonstationary policy: $\pi(s)$ depends on time left.
2. **Absorbing state(s)**: agent eventually *dies* for any $\pi$ ⇒ expected utility of every state is finite.
3. **Discounting**: assuming $\gamma < 1$, $R(s) \leq R_{\text{max}}$
   \[
   U([s_0, \ldots s_{\infty}]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq R_{\text{max}}/(1 - \gamma)
   \]
   Smaller $\gamma$ ⇒ shorter horizon
4. **Maximize system again**. Infinitive sequence can be compared in terms of average reward obtained per time step.

**Observation**: optimal policy has constant gain after initial transient.

E.g., taxi driver's daily scheme cruising for passengers
Dynamic programming: the Bellman equation

Definition of utility of states leads to a simple relationship among utilities of neighboring states:

\[
\text{expected sum of rewards} = \text{current reward} + \gamma \times \text{expected sum of rewards after taking best action}
\]

Bellman equation (1957):

\[
U(s) = R(s) + \gamma \max_a \sum_{s'} U(s')T(s,a,s')
\]
Example Bellman equation

\[ U(1, 1) = -0.04 \]
\[ + \gamma \max \{0.8U(1, 2) + 0.1U(2, 1) + 0.1U(1, 1), \text{ (up)} \] 
\[ + 0.9U(1, 1) + 0.1U(1, 2), \text{ (left)} \]
\[ + 0.9U(1, 1) + 0.1U(2, 1), \text{ (down)} \]
\[ + 0.8U(2, 1) + 0.1U(1, 2) + 0.1U(1, 1) \} \text{ (right)} \]

One equation per state = \( n \) nonlinear equations (by the MAX operator) in \( n \) unknowns (the utility of the states).
Value iteration algorithm

The Bellman equation is the basic of the value iteration algorithm for solving MDPs.

**Idea:** Start with arbitrary utility values
Update to make them locally consistent with Bellman equation.
Everywhere locally consistent IMPLIES global optimality

Repeat for every $s$ simultaneously until no change.

$$U(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} U(s') T(s, a, s')$$ for all $s$
Convergence

Why does the value iteration algorithm reach a fix-point?

We need a way to measure distances between utility vectors. We use the **max-norm:**

\[ ||U|| = \max_s |U(s)| \]

so \[ ||U - V|| = \text{maximum difference between } U \text{ and } V \]
Convergence

Let $U^t$ and $U^{t+1}$ be successive approximations to the true utility $U$.

**Theorem:** For any two approximations $U^t$ and $V^t$

$$||U^{t+1} - V^{t+1}|| \leq \gamma ||U^t - V^t||$$

In words: any distinct approximations must get closer to each other so, in particular, any approximation must get closer to the true $U$ and value iteration converges to a unique, stable, optimal solution.
Convergence – error estimation

$U^{\pi_i}(s)$ is the utility obtained if $\pi_i$ is executed starting in $s$ and the policy loss $||U^t - U||$ is the most the agent can lose by executing $\pi_i$ instead of the optimal policy $\pi^*$. 

Theorem: if $||U^t - U|| < \epsilon$, then $||U^{\pi_i} - U|| < 2\epsilon \gamma/(1 - \gamma)$

In words, once the change in $U^t$ becomes small, we are almost done.

OBSERVATIONS:

• MEU policy using $U^t$ may be optimal long before convergence of values.
• Therefore, we can get an optimal policy even when the utility function estimate is inaccurate.
Convergence

We know that

- We can bound the error in the utility estimates if we stop after a finite number of iterations.
- We can bound the policy loss that results from executing the corresponding MEU policy.

All the results depend on discounting $\gamma < 1$. 
Policy iteration algorithm

Howard, 1960: search for optimal policy and utility values simultaneously

Algorithm:

\[ \pi \leftarrow \text{an arbitrary initial policy} \]
repeat until no change in \( \pi \)
compute utilities given \( \pi \)
update \( \pi \) as if utilities were correct
(i.e., local MEU)

How do we implement the policy-evaluation routine?
Policy iteration

To compute utilities given a fixed $\pi$ (value determination):

$$U(s) = R(s) + \gamma \sum_{s'} U(s')T(s, \pi(s), s')$$  for all $s$

In words, $n$ simultaneous linear equations (max operator was removed) in $n$ unknowns, solve in $O(n^3)$
Modified policy iteration

Policy iteration often converges in few iterations, but each is expensive.

Idea: use a few steps of value iteration (but with $\pi$ fixed) starting from the value function produced the last time to produce an approximate value determination step.

Often converges much faster than pure VI or PI.

Reinforcement learning algorithms operate by performing such updates based on the observed transitions made in an initially unknown environment
Partial observability

POMDP has **an observation model** $O(s, e)$ defining the probability that the agent obtains evidence $e$ when in state $s$.

Agent does not know which state it is in then makes no sense to talk about policy $\pi(s)$.

**Theorem (Astrom, 1965):** the optimal policy in a POMDP is a function $\pi(b)$ where $b$ is the **belief state** (probability distribution over states).

We can convert a POMDP into an MDP in belief-state space, where $T(b, a, b')$ is the probability that the new belief state is $b'$ given that the current belief state is $b$ and the agent does $a$, i.e., essentially a filtering update step.
Partial observability contd.

Solutions automatically include information-gathering behavior.

If there are \( n \) states, \( b \) is an \( n \)-dimensional real-valued vector THEN solving POMDPs is very (actually, \( \text{PSPACE}- \)) hard!

The real world is a POMDP (with initially unknown \( T \) and \( 0 \))
Comments

- **Markov decision processes** formally describe an environment for Reinforcement Learning (RL)

- **Almost all RL problems can be formalised as MDPs**, e.g. Optimal control primarily deals with continuous MDPs

- **Partially observable problems can be converted into MDPs**
Some MDP Tools

- Markov Decision Processes (MDP) Toolbox - MATLAB
  http://www7.inra.fr/mia/T/MDPtoolbox/
- jMarkov,
  http://copa.uniandes.edu.co/software/jmarkov/index.html
- SPUDD,
  https://cs.uwaterloo.ca/~jhoey/research/spudd/index.php
Sources of this Lecture