Markov Decision Processes (MDPs)

Reading:
Kaelbling et al. 1996 (see class website)
Announcements

- **Project:**
  - Poster session: Friday May 5\textsuperscript{th} 2-5pm, NSH Atrium
    - please arrive a little early to set up

- **FCEs!!!**
  - Please, please, please, please, please, please, please give us your feedback, it helps us improve the class! 😊
    - [http://www.cmu.edu/fce](http://www.cmu.edu/fce)
Discount Factors

People in economics and probabilistic decision-making do this all the time.

The “Discounted sum of future rewards” using discount factor $\gamma$ is $\gamma \in (0, 1)$

(reward now) + $\gamma$ (reward in 1 time step) + $\gamma^2$ (reward in 2 time steps) + $\gamma^3$ (reward in 3 time steps) + ...

: (infinite sum)

For example:

$20 + \gamma \cdot 20 + \gamma^2 \cdot 20 + \gamma^3 \cdot 20 + \ldots$

$= \frac{20}{1-\gamma} = \frac{20}{0.1} = 200$
The Academic Life

Define:

- $V_A = \text{Expected discounted future rewards starting in state A}$
- $V_B = \text{Expected discounted future rewards starting in state B}$
- $V_T = \text{Tenured Prof}$
- $V_S = \text{On the Street}$
- $V_D = \text{Dead}$

Assume Discount Factor $\gamma = 0.9$

How do we compute $V_A, V_B, V_T, V_S, V_D$ ?
Computing the Future Rewards of an Academic

Assume Discount Factor $\gamma = 0.9$

$V_B = 60 + \gamma [0.6 V_B + 0.2 V_T + 0.2 V_D]$

$V_S = 10 + \gamma [0.7 V_S + 0.3 V_D]$

$V_T = 400 + \gamma [0.3 V_D + 0.7 V_T]$

$V_D = 0$

$V_T = \frac{400}{1 - 0.7 \gamma}$
Joint Decision Space

Markov Decision Process (MDP) Representation:

- **State space:**
  - Joint state $x$ of entire system

- **Action space:**
  - Joint action $a = \{a_1, \ldots, a_n\}$ for all agents

- **Reward function:**
  - Total reward $R(x, a)$
    - sometimes reward can depend on action

- **Transition model:**
  - Dynamics of the entire system $P(x' \mid x, a)$
Policy

Policy: \( \pi(x) = a \)

At state \( x \), action \( a \) for all agents

\( \pi(x_0) = \) both peasants get wood

\( \pi(x_1) = \) one peasant builds barrack, other gets gold

\( \pi(x_2) = \) peasants get gold, footmen attack
Value of Policy

Value: $V_\pi(x)$

Expected long-term reward starting from $x$

$$V_\pi(x_0) = \mathbb{E}_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \gamma^4 R(x_4) + \cdots]$$

Future rewards discounted by $\gamma \in [0,1)$
Computing the value of a policy

\[ V_\pi(x_0) = E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \gamma^4 R(x_4) + \cdots] \]

- Discounted value of a state:
  - value of starting from \( x_0 \) and continuing with policy \( \pi \) from then on
  \[
  V_\pi(x_0) = E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \cdots] = E_\pi[\sum_{t=0}^{\infty} \gamma^t R(x_t)]
  \]

- A recursion!
  \[
  V_\pi(x_0) = E_\pi[R(x_0)] + E_\pi[\gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \cdots] = E_\pi[R(x_0)] + \gamma E_\pi[V_\pi(x_1)]
  \]

  e.g. associate prof.
  \[
  V_\pi(x_0) = R(x_0) + \gamma \sum_{x_1} P(x_1|x_0, \pi(x_0)) V_\pi(x_1)
  \]
Computing the value of a policy 1 – the matrix inversion approach

\[ V_\pi(x) = R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_\pi(x') \]

- Solve by simple matrix inversion:

\[ V_\pi = (I - \gamma P_\pi)^{-1} R \]

\[ P_\pi = (1 \times 1) \begin{pmatrix} p(x' \mid x, a) \end{pmatrix} \]

\[ R = (1 \times 1) \begin{pmatrix} 9.8 & -1000 \\ & 1 \end{pmatrix} \]

\[ V_\pi = (1 \times 1) \begin{pmatrix} V_\pi(x) \end{pmatrix} \]

\[ I \text{ give you } V_\pi \]

\( x \) setting: give me \( \Pi \)
Computing the value of a policy 2 – iteratively

(Value Iteration)

$$V_\pi(x) = R(x) + \gamma \sum_{x'} P(x' | x, a = \pi(x)) V_\pi(x')$$

- If you have $1,000,000$ states, inverting a $1,000,000 \times 1,000,000$ matrix is hard!
- Can solve using a simple convergent iterative approach:
  (a.k.a. dynamic programming)
  - Start with some guess $V_0$
  - Iteratively say:
    - $V_{t+1} = R + \gamma P_\pi V_t$
  - Stop when $\|V_{t+1} - V_t\|_\infty \leq \varepsilon$
    - means that $\|V_\pi - V_{t+1}\|_\infty \leq \varepsilon/(1-\gamma)$

$$\|V\|_\infty = \max_x |V(x)|$$
But we want to learn a Policy

- So far, told you how good a policy is…
- But how can we choose the best policy???

Suppose there was only one time step:
- world is about to end!!!
- select action that maximizes reward!

Choose \( \pi(x) = \text{argmax}_a R(x,a) \)

At state \( x \), action \( a \) for all agents

\( \pi(x_0) = \text{both peasants get wood} \)

\( \pi(x_1) = \text{one peasant builds barrack, other gets gold} \)

\( \pi(x_2) = \text{peasants get gold, footmen attack} \)
Another recursion!

- Two time steps: address tradeoff
  - good reward now
  - better reward in the future

\[ V(x_{t=0}) = \max_a R(x_{t=0}, a) \]

\[ \Pi(x_{t=1}) = \arg\max_a R(x_{t=1}, a) + \gamma \sum_{x_{t=0}} p(x_{t=0}|x_{t=1}, a) V(x_{t=0}) \]
Unrolling the recursion

Choose actions that lead to best value in the long run

Optimal value policy achieves optimal value $V^*$

$$V^*(x_0) = \max_{a_0} R(x_0, a_0) + \gamma E_{a_0} \left[ \max_{a_1} R(x_1, a_1) + \gamma^2 E_{a_1} \left[ \max_{a_2} R(x_2, a_2) + \gamma^3 \cdots \right] \right]$$

$$V^*(x_0) = \max_a R(x_0, a) + \gamma \mathbb{E}_a \left[ V^*(x_1) \right]$$

$$V^*(x_0) = \max_a R(x_0, a) + \gamma \sum_{x_1} p(x_1 | x_0, a) V^*(x_1)$$
Bellman equation

- Evaluating policy $\pi$:
  \[
  V_\pi(x) = R(x) + \gamma \sum_{x'} P(x' | x, a = \pi(x)) V_\pi(x')
  \]

- Computing the optimal value $V^*$ - Bellman equation
  \[
  V^*(x) = \max_a \left[ R(x, a) + \gamma \sum_{x'} P(x' | x, a) V^*(x') \right]
  \]
Optimal Long-term Plan

Optimal value function $V^*(x)$

$Q^*(x, a) = R(x, a) + \gamma \sum_{x'} P(x'|x, a)V^*(x')$

Optimal policy:

$\pi^*(x) = \arg \max_a Q^*(x, a)$

is the greedy policy w.r.t. $V^*$.
Interesting fact – Unique value

\[ V^*(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x'|x, a)V^*(x') \]

- **Slightly surprising fact**: There is only one \( V^* \) that solves Bellman equation!
  - there may be many optimal policies that achieve \( V^* \)
- **Surprising fact**: optimal policies are good everywhere!!!

\[ V_{\pi^*}(x) \geq V_{\pi}(x), \ \forall x, \ \forall \pi \]
Solving an MDP

Bellman equation is non-linear!!!

Many algorithms solve the Bellman equations:

- **Policy iteration** [Howard ‘60, Bellman ‘57]
- **Value iteration** [Bellman ‘57]
- **Linear programming** [Manne ‘60]
- ...
Value iteration (a.k.a. dynamic programming) – the simplest of all

\[ V^*(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x' | x, a) V^*(x') \]

- Start with some guess \( V_0 \)
- Iteratively say:
  \[ V_{t+1}(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x' | x, a) V_t(x') \]
- Stop when \( \|V_{t+1} - V_t\|_\infty \leq \varepsilon \)

\( \varepsilon \) means that \( \|V^* - V_{t+1}\|_\infty \leq \varepsilon/(1-\gamma) \)

\( V^*(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x' | x, a) V_0(x') \)
A simple example

You run a startup company.

In every state you must choose between Saving money or Advertising.

\[ \gamma = 0.9 \]
Let’s compute $V_t(x)$ for our example

$$V_{t+1}(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x'|x, a) V_t(x')$$

Value iteration
Let’s compute $V_t(x)$ for our example

\[ V_{t+1}(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x' | x, a) V_t(x') \]

<table>
<thead>
<tr>
<th>$t$</th>
<th>$V_t(\text{PU})$</th>
<th>$V_t(\text{PF})$</th>
<th>$V_t(\text{RU})$</th>
<th>$V_t(\text{RF})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4.5</td>
<td>14.5</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>2.03</td>
<td>6.53</td>
<td>25.08</td>
<td>18.55</td>
</tr>
<tr>
<td>4</td>
<td>3.852</td>
<td>12.20</td>
<td>29.63</td>
<td>19.26</td>
</tr>
<tr>
<td>5</td>
<td>7.22</td>
<td>15.07</td>
<td>32.00</td>
<td>20.40</td>
</tr>
<tr>
<td>6</td>
<td>10.03</td>
<td>17.65</td>
<td>33.58</td>
<td>22.43</td>
</tr>
</tbody>
</table>

$\gamma = 0.9$
Policy iteration – Another approach for computing \( \pi^* \)

- Start with some guess for a policy \( \pi_0 \)
- Iteratively say:
  - evaluate policy: \( V_t(x) = R(x, a = \pi_t(x)) + \gamma \sum_{x'} P(x'|x, a = \pi_t(x))V_t(x') \)
  - greedily improve policy: \( \pi_{t+1}(x) = \arg \max_a R(x, a) + \gamma \sum_{x'} P(x'|x, a)V_t(x') \)

- Stop when
  - policy stops changing
    - usually happens in about 10 iterations
  - or \( \|V_{t+1} - V_t\|_\infty \leq \varepsilon \)
    - means that \( \|V^* - V_{t+1}\|_\infty \leq \varepsilon/(1-\gamma) \)

Open problem: how long will policy iteration take?
I think largest known lower bound is \( \mathcal{O}(n^{1.5}) \) for \( n \) states.
Policy Iteration & Value Iteration: Which is best ???

It depends.

- Lots of actions? Choose Policy Iteration
- Already got a fair policy? Policy Iteration
- Few actions, acyclic? Value Iteration

Best of Both Worlds:

- Modified Policy Iteration [Puterman]
  
  ...a simple mix of value iteration and policy iteration
  
  use iterative approach instead of matrix inversion to evaluate a policy.

3rd Approach

Linear Programming
LP Solution to MDP

Value computed by linear programming:

\[
\text{minimize: } \sum_{x} V(x) \quad \text{variables in LP are } V(x) \text{ in } n \text{ variables}
\]

\[
V(x) = \max_a R(x, a) + \delta \sum_{x'} P(x'|x, a)V(x')
\]

subject to:

\[
\begin{align*}
V(x) &\geq R(x, a) + \gamma \sum_{x'} P(x'|x, a)V(x') \\
\forall x, a
\end{align*}
\]

- One variable \( V(x) \) for each state
- One constraint for each state \( x \) and action \( a \)
- **Polynomial time solution**

Poly Nat 3 constraints are polynomial in input

\[ \Rightarrow \text{ MDPs are in } P \]
What you need to know

- What’s a Markov decision process
  - state, actions, transitions, rewards
  - a policy
  - value function for a policy
    - computing $V_\pi$
- Optimal value function and optimal policy
  - Bellman equation
- Solving Bellman equation
  - with value iteration, policy iteration and linear programming
Acknowledgment

- This lecture contains some material from Andrew Moore’s excellent collection of ML tutorials:
  - [http://www.cs.cmu.edu/~awm/tutorials](http://www.cs.cmu.edu/~awm/tutorials)
Reinforcement Learning

Machine Learning – 10701/15781
Carlos Guestrin
Carnegie Mellon University
May 1st, 2006
The Reinforcement Learning task

**World:** You are in state 34.
Your immediate reward is 3. You have possible 3 actions.

**Robot:** I’ll take action 2.

**World:** You are in state 77.
Your immediate reward is -7. You have possible 2 actions.

**Robot:** I’ll take action 1.

**World:** You’re in state 34 (again).
Your immediate reward is 3. You have possible 3 actions.
Formalizing the (online) reinforcement learning problem

- Given a set of states $X$ and actions $A$
  - in some versions of the problem size of $X$ and $A$ unknown

- Interact with world at each time step $t$:
  - world gives state $x_t$ and reward $r_t$
  - you give next action $a_t$

- Goal: (quickly) learn policy that (approximately) maximizes long-term expected discounted reward
The "Credit Assignment" Problem

I’m in state 43, reward = 0, action = 2

- 39, " = 0, " = 4
- 22, " = 0, " = 1
- 21, " = 0, " = 1
- 21, " = 0, " = 1
- 13, " = 0, " = 2
- 54, " = 0, " = 2
- 26, " = 100,

Yippee! I got to a state with a big reward! But which of my actions along the way actually helped me get there??

This is the Credit Assignment problem.
Exploration-Exploitation tradeoff

- You have visited part of the state space and found a reward of 100
  - is this the best I can hope for???

- **Exploitation**: should I stick with what I know and find a good policy w.r.t. this knowledge?
  - at the risk of missing out on some large reward somewhere

- **Exploration**: should I look for a region with more reward?
  - at the risk of wasting my time or collecting a lot of negative reward
Two main reinforcement learning approaches

- **Model-based approaches:**
  - explore environment → learn model \( P(x'|x,a) \) and \( R(x,a) \) (almost) everywhere
  - use model to plan policy, MDP-style
  - approach leads to strongest theoretical results
  - works quite well in practice when state space is manageable

- **Model-free approach:**
  - don’t learn a model → learn value function or policy directly
  - leads to weaker theoretical results
  - often works well when state space is large
Rmax – A model-based approach

Brafman & Tennenholtz 2002
(see class website)
Given a dataset – learn model

Given data, learn (MDP) Representation:

- Dataset:

- Learn reward function:
  - $R(x,a)$

- Learn transition model:
  - $P(x'|x,a)$
Some challenges in model-based RL 1: Planning with insufficient information

- Model-based approach:
  - estimate $R(x,a) \& P(x'|x,a)$
  - obtain policy by value or policy iteration, or linear programming
  - No credit assignment problem $\rightarrow$ learning model, planning algorithm takes care of “assigning” credit

- What do you plug in when you don’t have enough information about a state?
  - don’t reward at a particular state
    - plug in smallest reward ($R_{\text{min}}$)?
    - plug in largest reward ($R_{\text{max}}$)?
  - don’t know a particular transition probability?
Some challenges in model-based RL 2: Exploration-Exploitation tradeoff

- A state may be very hard to reach
  - waste a lot of time trying to learn rewards and transitions for this state
  - after a much effort, state may be useless

- A strong advantage of a model-based approach:
  - you know which states estimate for rewards and transitions are bad
  - can (try) to plan to reach these states
  - have a good estimate of how long it takes to get there
A surprisingly simple approach for model based RL – The Rmax algorithm [Brafman & Tennenholtz]

- **Optimism in the face of uncertainty!!!!**
  - heuristic shown to be useful long before theory was done (e.g., Kaelbling ’90)

- If you don’t know reward for a particular state-action pair, set it to $R_{\text{max}}$!!!

- If you don’t know the transition probabilities $P(x'|x,a)$ from some state action pair $x,a$ assume you go to a magic, fairytale new state $x_0$!!!
  - $R(x_0,a) = R_{\text{max}}$
  - $P(x_0|x_0,a) = 1$
Understanding $R_{\text{max}}$

- With $R_{\text{max}}$ you either:
  - **explore** – visit a state-action pair you don’t know much about
    - because it seems to have lots of potential
  - **exploit** – spend all your time on known states
    - even if unknown states were amazingly good, it’s not worth it

- Note: you never know if you are exploring or exploiting!!!
Implicit Exploration-Exploitation Lemma

**Lemma**: every T time steps, either:

- **Exploits**: achieves near-optimal reward for these T-steps, or
- **Explores**: with high probability, the agent visits an unknown state-action pair
  - learns a little about an unknown state
- T is related to *mixing time* of Markov chain defined by MDP
  - time it takes to (approximately) forget where you started
The Rmax algorithm

- **Initialization:**
  - Add state $x_0$ to MDP
  - $R(x,a) = R_{\text{max}}, \forall x,a$
  - $P(x_0|x,a) = 1, \forall x,a$
  - all states (except for $x_0$) are **unknown**

- **Repeat**
  - obtain policy for current MDP and Execute policy
  - for any visited state-action pair, set reward function to appropriate value
  - if visited some state-action pair $x,a$ enough times to estimate $P(x'|x,a)$
    - update transition probs. $P(x'|x,a)$ for $x,a$ using MLE
    - recompute policy
Visit enough times to estimate $P(x'|x,a)$?

- How many times are enough?
  - use Chernoff Bound!

**Chernoff Bound:**
- $X_1,\ldots,X_n$ are i.i.d. Bernoulli trials with prob. $\theta$
- $P(|1/n \sum_i X_i - \theta| > \varepsilon) \leq \exp\{-2n\varepsilon^2\}$
Putting it all together

**Theorem:** With prob. at least $1-\delta$, Rmax will reach a $\varepsilon$-optimal policy in time polynomial in: num. states, num. actions, $T$, $1/\varepsilon$, $1/\delta$

- Every $T$ steps:
  - achieve near optimal reward (great!), or
  - visit an unknown state-action pair $\rightarrow$ num. states and actions is finite, so can’t take too long before all states are known
Problems with model-based approach

- If state space is large
  - transition matrix is very large!
  - requires many visits to declare a state as known

- Hard to do “approximate” learning with large state spaces
  - some options exist, though
TD-Learning and Q-learning – Model-free approaches
Value of Policy

Value: $V_\pi(x)$

Expected long-term reward starting from $x$

$$V_\pi(x_0) = E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \gamma^4 R(x_4) + \cdots]$$

Future rewards discounted by $\gamma \in [0,1)$
A simple monte-carlo policy evaluation

- Estimate $V(x)$, start several trajectories from $x \rightarrow V(x)$ is average reward from these trajectories
  - Hoeffding’s inequality tells you how many you need
  - discounted reward $\rightarrow$ don’t have to run each trajectory forever to get reward estimate
Problems with monte-carlo approach

- **Resets**: assumes you can restart process from same state many times

- **Wasteful**: same trajectory can be used to estimate many states
Reusing trajectories

- Value determination:
  \[ V_\pi(x) = R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_\pi(x') \]

- Expressed as an expectation over next states:
  \[ V_\pi(x) = R(x) + \gamma E \left[ V_\pi(x') \mid x, a = \pi(x) \right] \]

- Initialize value function (zeros, at random,...)
- Idea 1: Observe a transition: \( x_t \to x_{t+1}, r_{t+1} \), approximate expec. with single sample:
  - unbiased!!
  - but a very bad estimate!!!
Simple fix: Temporal Difference (TD) Learning

- Idea 2: Observe a transition: \( x_t \rightarrow x_{t+1}, r_{t+1} \), approximate expec. by mixture of new sample with old estimate:

- \( \alpha > 0 \) is learning rate
TD converges (can take a long time!!!)

\[ V_\pi(x) = R(x) + \gamma \sum_{x'} P(x' | x, a = \pi(x)) V_\pi(x') \]

- **Theorem**: TD converges in the limit (with prob. 1), if:
  - every state is visited infinitely often
  - Learning rate decays just so:
    - \( \sum_{i=1}^{\infty} \alpha_i = \infty \)
    - \( \sum_{i=1}^{\infty} \alpha_i^2 < \infty \)