Markov Decision Processes (MDPs)

Machine Learning – CSE546
Carlos Guestrin
University of Washington
December 2, 2013

©Carlos Guestrin 2005-2013

Reinforcement Learning

training by feedback
Learning to act

- Reinforcement learning
- An agent
  - Makes sensor observations
  - Must select action
  - Receives rewards
    - positive for "good" states
    - negative for "bad" states

[Ng et al. '05]

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 $P(x'|x,a)$
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

$$\text{(reward now)} + \gamma \text{(reward in 1 time step)} + \gamma^2 \text{(reward in 2 time steps)} + \gamma^3 \text{(reward in 3 time steps)} + \cdots$$

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 $V_A$, $V_B$, $V_T$, $V_S$, $V_D$?
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 \( x \)

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

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

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

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

Simple approach for computing the value of a policy: Iteratively

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

- Can solve using a simple convergent iterative approach:
  (a.k.a. dynamic programming)
  - Start with some guess \( V^0 \)
  - Iteratively say:
    \[ V^{t+1}_\pi(x) \leftarrow R(x) + \gamma \sum_{x'} P(x' | x, a = \pi(x)) V^t_\pi(x') \]
  - Stop when \( ||V^{t+1}_\pi - V^t_\pi||_\infty < \epsilon \)
    - means that \( ||V^t_\pi - V^{t+1}_\pi||_\infty < \epsilon/(1-\gamma) \)
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!

Unrolling the recursion

- Choose actions that lead to best value in the long run
  - Optimal value policy achieves optimal value $V^*$

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

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

- Computing the optimal value $V^*$ - Bellman equation
  \[ V^*(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x' \mid x, a) V^*(x') \]

Optimal Long-term Plan

Optimal value function $V^*(x)$  \[ \rightarrow \]  Optimal Policy: $\pi^*(x)$

Optimal policy:
\[ \pi^*(x) = \arg\max_a R(x, a) + \gamma \sum_{x'} P(x' \mid x, a) V^*(x') \]
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

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

Optimal value \( V^*(x) \)

Optimal policy \( \pi^*(x) \)

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

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

- Start with some guess \( V^0 \)
- Iteratively say:
  \[ V^{t+1}(x) \leftarrow \max_a R(x, a) + \gamma \sum_{x'} P(x' \mid x, a) V^t(x') \]
- Stop when \( \| V_{t+1} - V_t \|_\infty < \epsilon \)
  - means that \( \| V^* - V_{t+1} \|_\infty < t/(1-\gamma) \)

A simple example

You run a startup company.
In every state you must choose between Saving money or Advertising.

©Carlos Guestrin 2005-2013
Let's compute $V_t(x)$ for our example

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

<table>
<thead>
<tr>
<th>$t$</th>
<th>$V'(PU)$</th>
<th>$V'(PF)$</th>
<th>$V'(RU)$</th>
<th>$V'(RF)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\infty$</td>
<td>31.59</td>
<td>38.60</td>
<td>44.02</td>
<td>54.02</td>
</tr>
</tbody>
</table>
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