Markov Decision Processes (MDPs)

Machine Learning – CSEP546
Carlos Guestrin
University of Washington
March 10, 2014
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]

©Carlos Guestrin 2005-2014
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 $P(x'|x,a)$
People in economics and probabilistic decision-making do this all the time.

The “Discounted sum of future rewards” using discount factor $\gamma$ is

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

(infinite sum)
Define:

\[ V_A = \text{Expected discounted future rewards starting in state A} \]
\[ V_B = \text{Expected discounted future rewards starting in state B} \]
\[ V_T = \text{Expected discounted future rewards starting in state T} \]
\[ V_S = \text{Expected discounted future rewards starting in state S} \]
\[ V_D = \text{Expected discounted future rewards starting in state D} \]

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

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

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

$\pi(x_2) = \text{peasants get gold, footmen attack}$
Value of Policy

Value: $V_\pi(x)$

Expected long-term reward starting from $x$

$V_\pi(x_0) = E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \gamma^4 R(x_4) + \ldots]$
Computing the value of a policy

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

- Discounted value of a state:
  - value of starting from \( x_0 \) and continuing with policy \( \pi \) from then on

\[ V_\pi(x_0) = E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \ldots] \]
\[ = E_\pi[\sum_{t=0}^{\infty} \gamma^t R(x_t)] \]

- A recursion!
Simple approach for computing the value of a policy: Iteratively

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

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

---

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

©Carlos Guestrin 2005-2014
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 $V^*(x)$

Optimal Policy: $\pi^*(x)$

Optimal policy:

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

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 \( \|V^{t+1} - V^t\|_\infty < \varepsilon \)
  - \( \square \) means that \( \|V^* - V^{t+1}\|_\infty < \varepsilon/(1-\gamma) \)
A simple example

You run a startup company.
In every state you must choose between Saving money or Advertising.
Let’s compute $V_t(x)$ for our example

$$V^{t+1}(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x' | x, a) V^t(x')$$
Let’s compute $V_t(x)$ for our example

$$V^{t+1}(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x' | x, a) V^t(x')$$
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
Reinforcement Learning

Machine Learning – CSE546
Carlos Guestrin
University of Washington

December 3, 2013
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

©Carlos Guestrin 2005-2014
Two main reinforcement learning approaches

- Model-based approaches:
  - explore environment, then 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
Given a dataset – learn model

Given data, learn (MDP) Representation:

- Dataset:
- Learn reward function:
  - $R(x,a)$
- Learn transition model:
  - $P(x'|x,a)$
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!
    - 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 0?
    - 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 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 an $\epsilon$-optimal policy in time polynomial in: num. states, num. actions, $T$, $1/\epsilon$, $1/\delta$

- Every $T$ steps:
  - achieve near optimal reward (great!), or
  - visit an unknown state-action pair! num. states and actions is finite, so can’t take too long before all states are known
What you need to know about RL…

- Neither supervised, nor unsupervised learning
- Try to learn to act in the world, as we travel states and get rewards
- Model-based & Model-free approaches
- Rmax, a model based approach:
  - Learn model of rewards and transitions
  - Address exploration-exploitation tradeoff
  - Simple algorithm, great in practice
Announcement

- **Poster session March 17th 6:30pm CSE Atrium:**
  - Arrive on time to set up!
  - We will provide poster boards and pins (see website)
  - Everyone must attend
  - Write project number on poster
    - (we will send these out this week)
  - Prepare 2-3 minutes overview of what you did
  - At least 2 instructors will see your project

- **Final Project Report**
  - Due March 19th at 6:30pm – no late days!
    - We have to send out your grades
  - See website for details (maximum 8 pages)
  - Be clear about what you did
  - Make it read like a paper