Markov Decision Processes (MDPs)

Machine Learning – CSE546
Carlos Guestrin
University of Washington

December 2, 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]

Markov Decision Process (MDP) Representation

- State space:
  - Joint state $\mathbf{x}$ of entire system
- Action space:
  - Joint action $\mathbf{a} = \{a_1, \ldots, a_n\}$ for all agents
- Reward function:
  - Total reward $R(\mathbf{x}, \mathbf{a})$
    - sometimes reward can depend on action
- Transition model:
  - Dynamics of the entire system $P(\mathbf{x}' | \mathbf{x}, \mathbf{a})$

Discount Factors

$0 < \gamma < 1$

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

The "Discounted sum of future rewards" using discount factor γ" is

(reward now) +

$R_1 + \gamma R_2 + \gamma^2 R_3 \ldots$

γ (reward in 1 time step) +

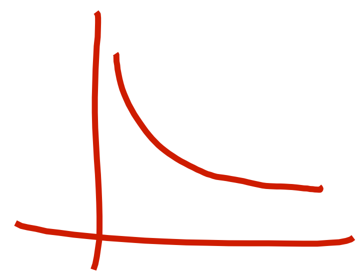
γ^2 (reward in 2 time steps) +

γ^3 (reward in 3 time steps) +

:

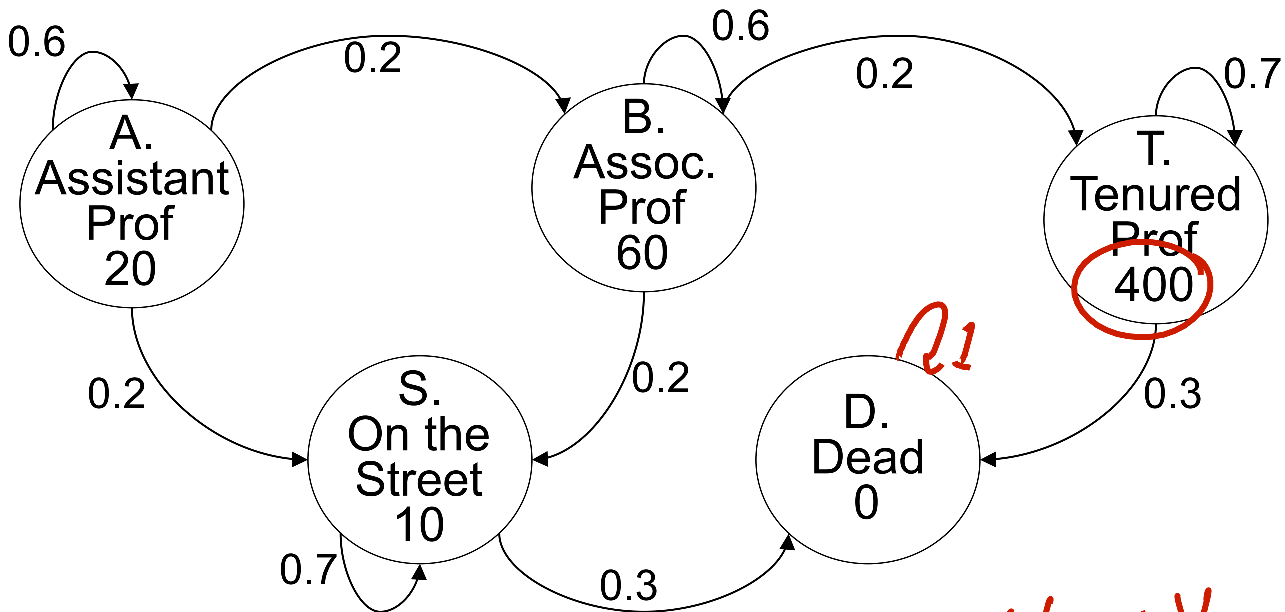
: (infinite sum)

Rewards in future are worth exponentially less



The Academic Life

Assume Discount Factor $\gamma = 0.9$



Define:

V_A = Expected discounted future rewards starting in state A

V_B = Expected discounted future rewards starting in state B

V_T = " " " " " " " T

V_S = " " " " " " " S

V_D = " " " " " " " D

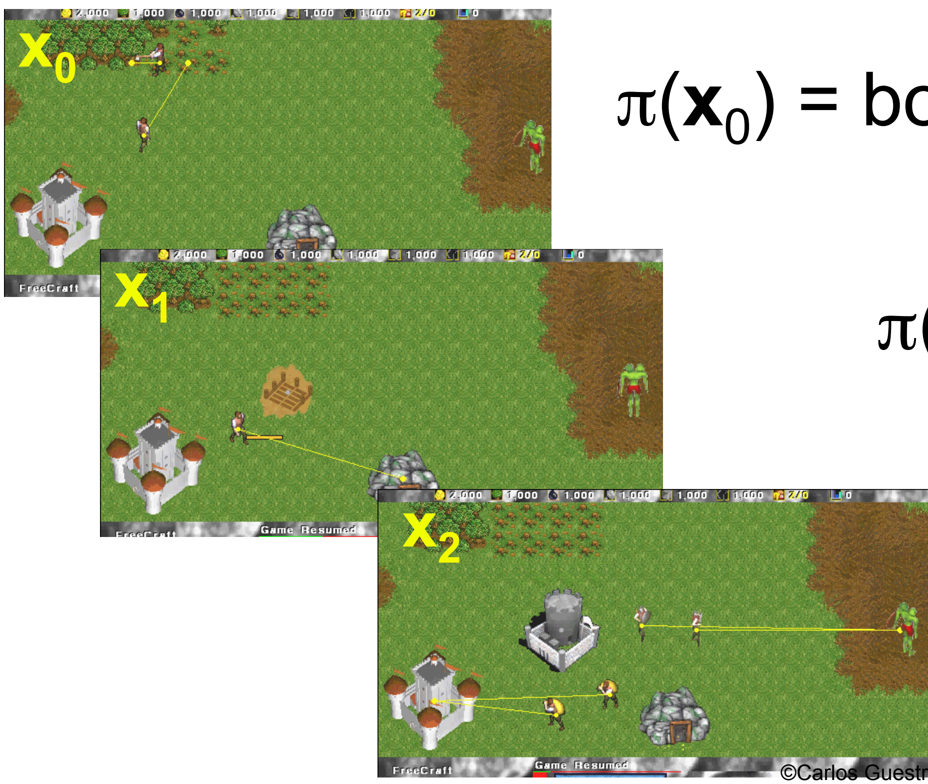
How do we compute V_A, V_B, V_T, V_S, V_D ?

$V_A = 20 + \gamma(0.6 V_A + 0.2 V_B + 0.2 V_S)$

Policy

Policy: $\pi(\mathbf{x}) = \mathbf{a}$

At state $\mathbf{x}$, action $\mathbf{a}$ for all agents



$\pi(\mathbf{x}_0)$ = both peasants get wood

$\pi(\mathbf{x}_1)$ = one peasant builds barrack, other gets gold

$\pi(\mathbf{x}_2)$ = peasants get gold, footmen attack

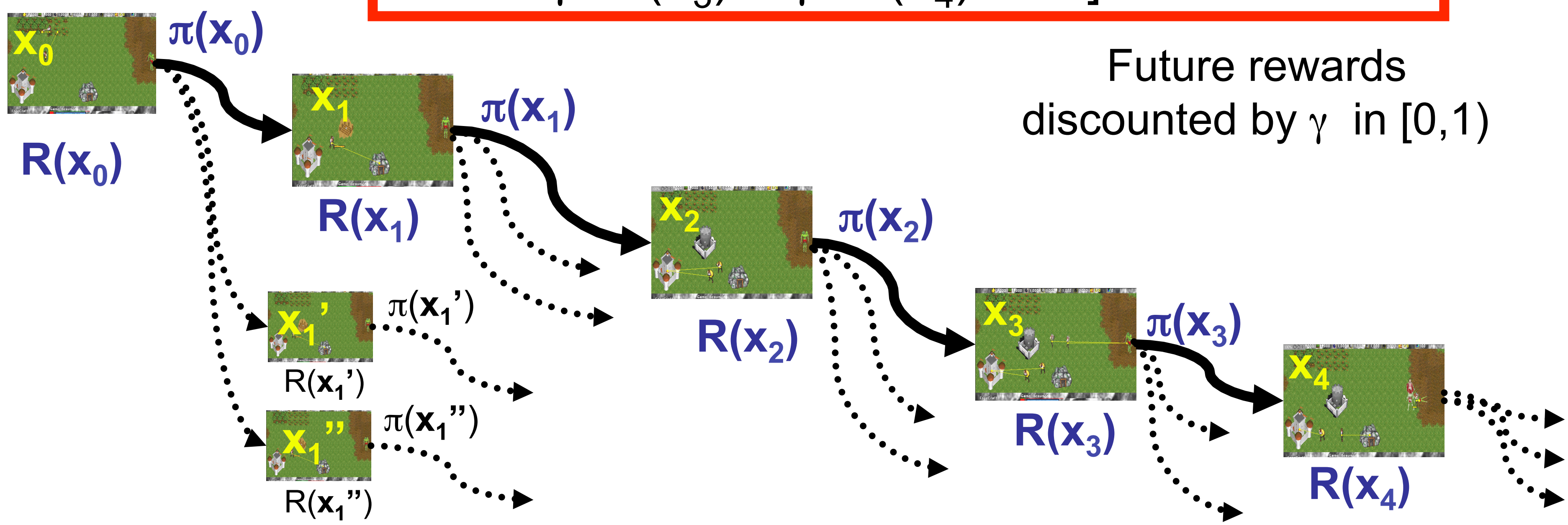
Value of Policy

Value: $V_\pi(\mathbf{x})$

Expected long-term reward starting from **x**

$$V_\pi(\mathbf{x_0}) = \mathbf{E}_\pi[R(\mathbf{x}_0) + \gamma\, R(\mathbf{x}_1) + \gamma^2\, R(\mathbf{x}_2) + \gamma^3\, R(\mathbf{x}_3) + \gamma^4\, R(\mathbf{x}_4) + \ldots]$$

Start from $\mathbf{x_0}$



Future rewards discounted by γ in [0,1)

Computing the value of a policy

$$V_\pi(\mathbf{x_0}) = \mathbf{E}_\pi[R(\mathbf{x}_0) + \gamma\, R(\mathbf{x}_1) + \gamma^2\, R(\mathbf{x}_2) + \gamma^3\, R(\mathbf{x}_3) + \gamma^4\, R(\mathbf{x}_4) + \ldots]$$

- Discounted value of a state:
  - value of starting from x_0 and continuing with policy π from then on

$$\begin{aligned} V_\pi(x_0) &= E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \cdots] \\ &= E_\pi[\sum_{t=0}^{\infty} \gamma^t R(x_t)] \end{aligned}$$

- A recursion!

Simple approach for computing the value of a policy: Iteratively

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

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

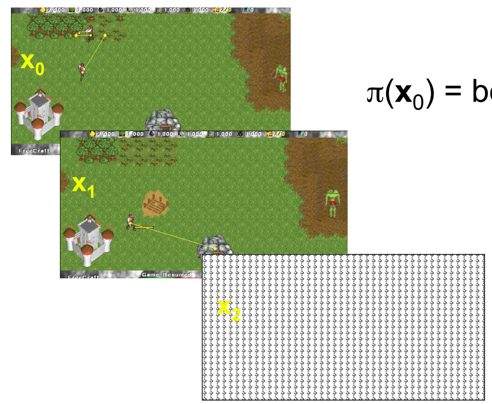
But we want to learn a **Policy**

- 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: $\pi(\mathbf{x}) = \mathbf{a}$

At state $\mathbf{x}$, action $\mathbf{a}$ for all agents



$\pi(\mathbf{x}_0)$ = both peasants get wood

$\pi(\mathbf{x}_1)$ = one peasant builds barrack, other gets gold

$\pi(\mathbf{x}_2)$ = peasants get gold, footmen attack

Unrolling the recursion

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

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

Bellman equation

- Evaluating policy π:

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

- Computing the optimal value V* - Bellman equation

$$V^*(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) V^*(\mathbf{x}')$$

Interesting fact – Unique value

$$V^*(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^*(\mathbf{x}')$$

- *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 $\Rightarrow$ Optimal value $V^*(\mathbf{x})$ $\Rightarrow$ Optimal policy $\pi^*(\mathbf{x})$

$$V^*(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^*(\mathbf{x}')$$

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

Optimal Long-term Plan

Optimal value function $V^*(\mathbf{x})$

Optimal Policy: $\pi^*(\mathbf{x})$

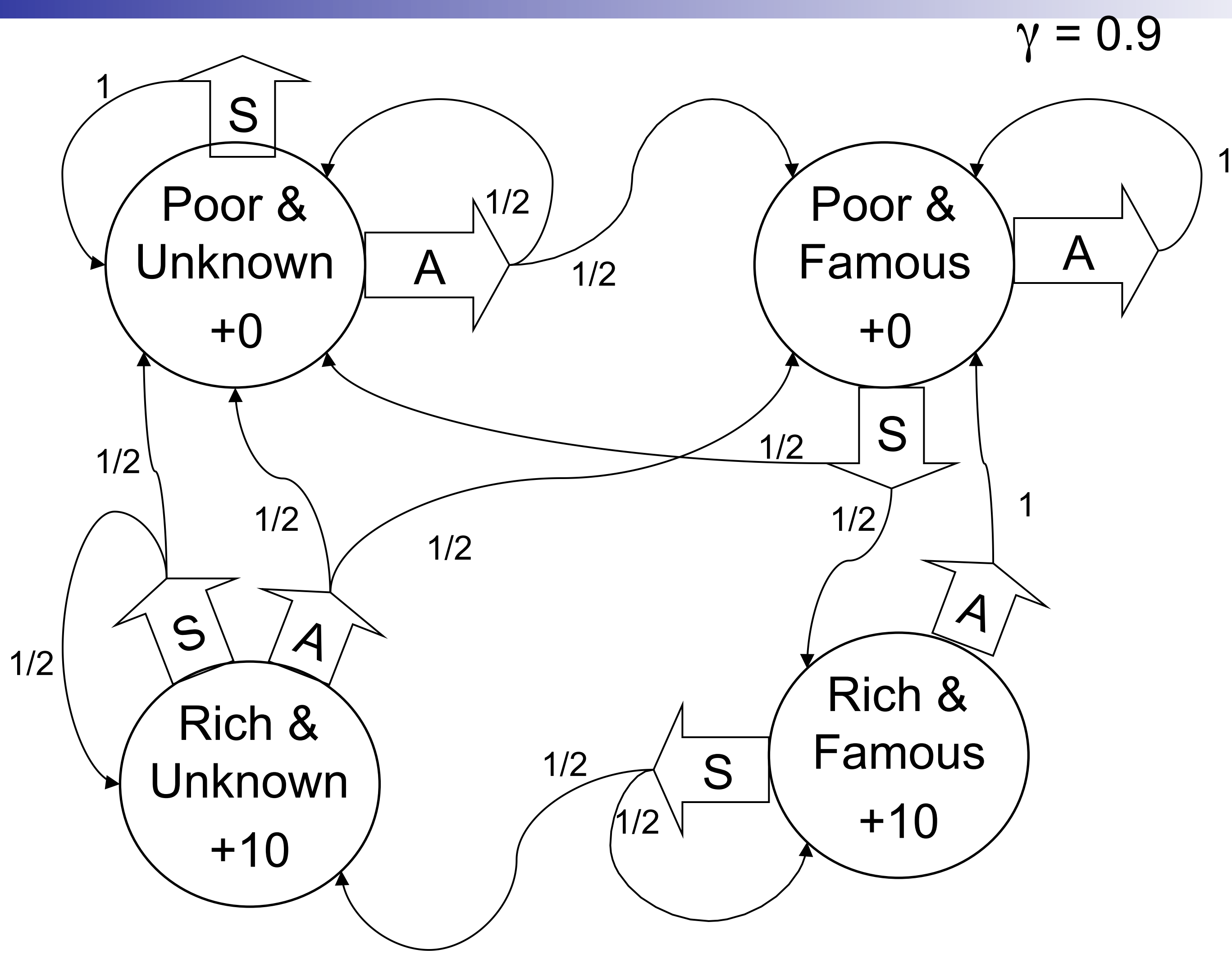
Optimal policy:

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

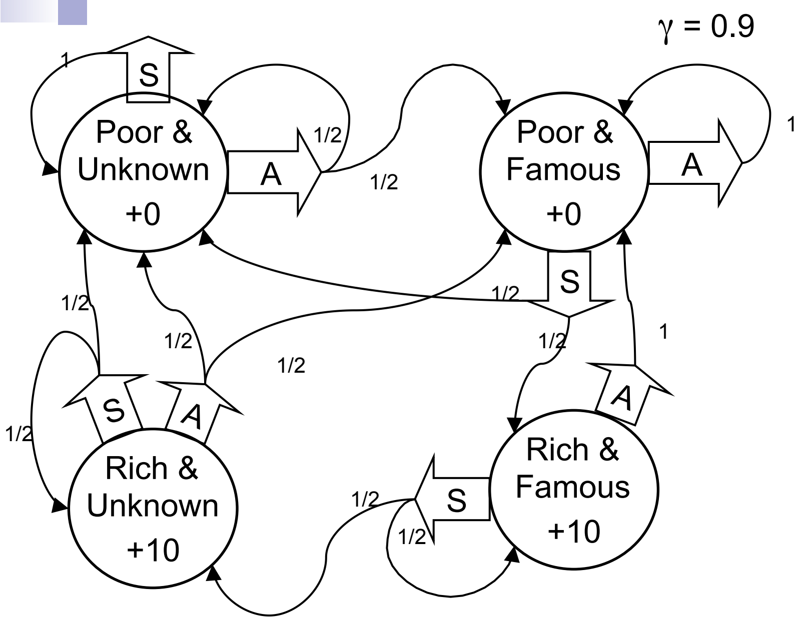
A simple example

You run a startup company.

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



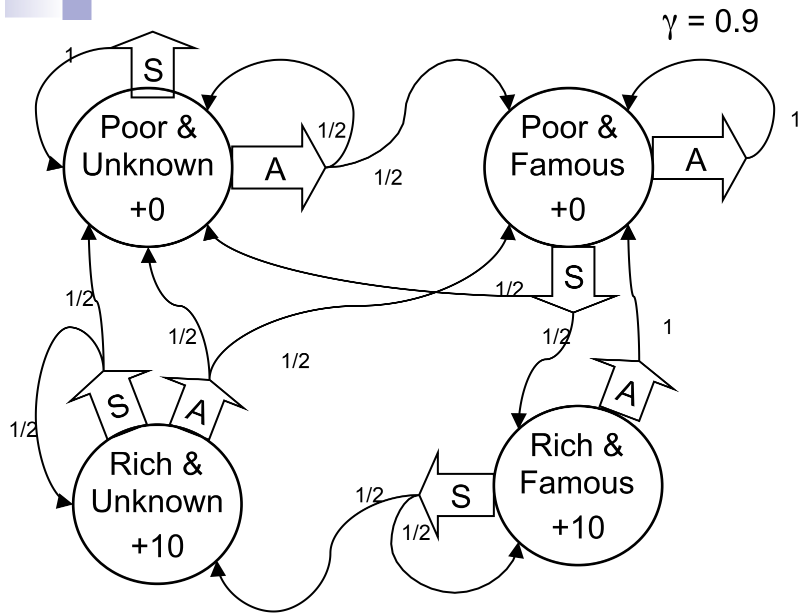
Let's compute $V_t(x)$ for our example



t	V^t(PU)	V^t(PF)	V^t(RU)	V^t(RF)
1				
2				
3				
4				
5				
6				

$$V^{t+1}(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^t(\mathbf{x}')$$

Let's compute $V_t(x)$ for our example



t	V^t(PU)	V^t(PF)	V^t(RU)	V^t(RF)
1	0	0	10	10
2	0	4.5	14.5	19
3	2.03	9.46	17.44	25.08
4	5.17	13.61	20.17	29.13
5	8.45	16.91	22.88	32.19
6	11.41	19.62	25.43	34.78
∞	31.59	38.60	44.02	54.02

$$V^{t+1}(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^t(\mathbf{x}')$$

What you need to know

- What's a Markov decision process
  - state, actions, transitions, rewards
  - a policy
  - value function for a policy
    - computing V_π
- 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 4, 2014

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 $\mathbf{x}_t$ and reward r_t
  - you give next action $\mathbf{a}_t$
- **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, 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 $R(\mathbf{x},\mathbf{a})$ & $P(\mathbf{x}'|\mathbf{x},\mathbf{a})$
  - 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_{min})?
    - plug in largest reward (R_{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

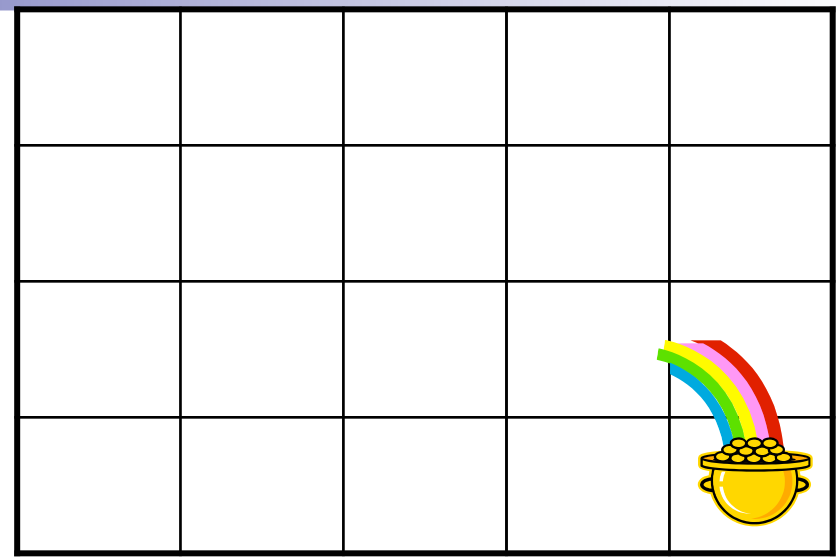
A surprisingly simple approach for model based RL – The Rmax algorithm [Brafman & Tennenholtz]

- **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_{max}!!!

- If you don't know the transition probabilities $P(\mathbf{x}'|\mathbf{x},\mathbf{a})$ from some some state action pair $\mathbf{x},\mathbf{a}$ assume you go to **a magic, fairytale** new state $\mathbf{x}_0$!!!
  - $R(\mathbf{x}_0,\mathbf{a}) = R_{max}$
  - $P(\mathbf{x_0}|\mathbf{x_0},\mathbf{a}) = 1$

Understanding R_{max}

- With R_{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 $\mathbf{x}_0$ to MDP
  - $R(\mathbf{x},\mathbf{a}) = R_{max}$, $\forall \mathbf{x},\mathbf{a}$
  - $P(\mathbf{x_0}|\mathbf{x},\mathbf{a}) = 1$, $\forall \mathbf{x},\mathbf{a}$
  - all states (except for $\mathbf{x}_0$) 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 $P(\mathbf{x'}|\mathbf{x},\mathbf{a})$
    - update transition probs. $P(\mathbf{x'}|\mathbf{x},\mathbf{a})$ 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. θ
  - $P(|1/n \sum_i X_i - \theta| > \varepsilon) \leq \exp\{-2n\varepsilon^2\}$

Putting it all together

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