

# Markov Decision Processes (MDPs) (cont.)

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

November 29<sup>th</sup>, 2007

1

## Markov Decision Process (MDP) Representation

- State space:
  - Joint state  $\mathbf{x}$  of entire system
- Action space:
  - Joint action  $\mathbf{a} = \{a_1, \dots, a_n\}$  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  $P(\mathbf{x}' | \mathbf{x}, \mathbf{a})$



# Computing the value of a policy

linearity of expectations  
 $E[a+b] = E[a] + E[b]$

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

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

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

- A recursion!

$$\begin{aligned} V_{\pi}(x_0) &= E_{\pi}[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \dots] \\ &= E_{\pi}[R(x_0)] + \gamma E_{\pi}[R(x_1) + \gamma R(x_2) + \gamma^2 R(x_3) + \dots] \\ &= R(x_0) + \gamma \sum_{x_1} P(x_1 | x_0, \pi(x_0)) V_{\pi}(x_1) \end{aligned}$$

# 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  $V_0$

any guess works, but a good guess is  $V_0(x) = R(x_0)$

- Iteratively say:

- $V_{t+1} = R + \gamma P_{\pi} V_t$

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

- Stop when  $\|V_{t+1} - V_t\| \leq \epsilon$

- means that  $\|V_{\pi} - V_{t+1}\| \leq \epsilon / (1 - \gamma)$

# But we want to learn a Policy

■ So far, told you how good a policy is...  $V_{\pi}(x)$

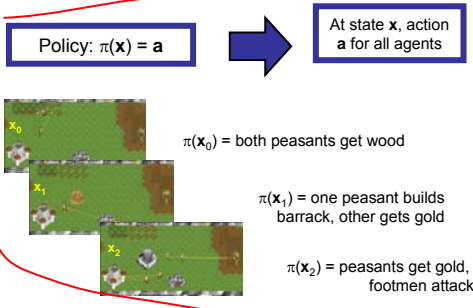
■ But how can we choose the best policy???

■ Suppose there was only one time step:

- world is about to end!!!
- select action that maximizes reward!

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

*for state x*



# Unrolling the recursion

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

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

*OPT Value*

$$V^*(x_0) = \max_{a_0} R(x_0, a_0) + \gamma \sum_{x_1} P(x_1 | x_0, a_0) V^*(x_1)$$

*Bellman Eqn.*

# Bellman equation

- Evaluating policy  $\pi$ :

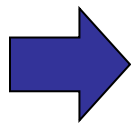
$$V_{\pi}(x) = R(x) + \gamma \sum_{x'} P(x' | x, a = \pi(x)) V_{\pi}(x')$$

- Computing the optimal value  $V^*$  - Bellman equation

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

# Optimal Long-term Plan

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



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

**Optimal policy:**

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

if I have  $V^*$ , then OPT policy is Greedy, but Greedy wrt  $V^*(x')$

## Interesting fact – Unique value

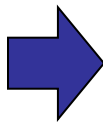
$$V^*(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' | \mathbf{x}, \mathbf{a}) V^*(\mathbf{x}')$$

- 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!!!

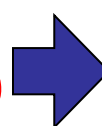
$$\underline{V_{\pi^*}(x)} \geq V_{\pi}(x), \quad \underline{\forall x}, \quad \underline{\forall \pi}$$

## Solving an MDP

Solve  
Bellman  
equation



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



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

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

**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

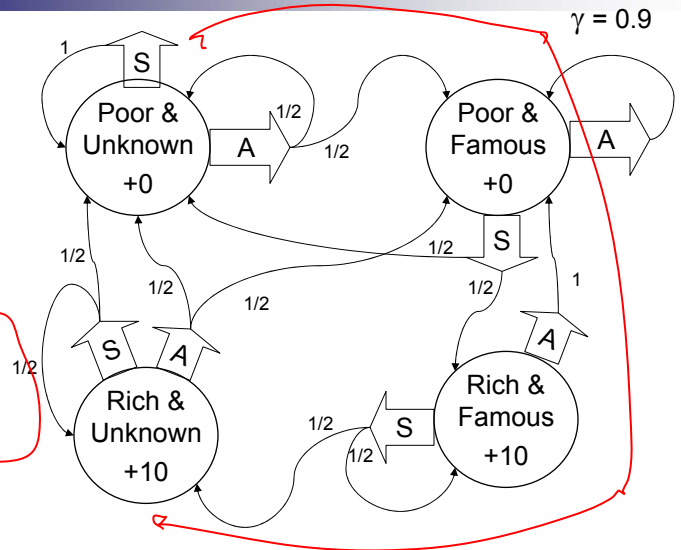
$$V^*(\mathbf{x}) = \max_a R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' | \mathbf{x}, \mathbf{a}) V^*(\mathbf{x}')$$

- Start with some guess  $V_0$  ← eg.,  $V_0^*(x) = R(x)$
- Iteratively say: then max over a compute for each a
  - $V_{t+1}(\mathbf{x}) = \max_a R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' | \mathbf{x}, \mathbf{a}) V_t(\mathbf{x}')$
- Stop when  $\|V_{t+1} - V_t\|_0 \leq \epsilon$ 
  - means that  $\|V^* - V_{t+1}\|_0 \leq \epsilon / (1 - \gamma)$

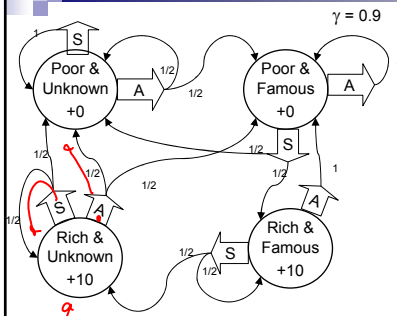
# A simple example

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



t	$V_t(\text{PU})$	$V_t(\text{PF})$	$V_t(\text{RU})$	$V_t(\text{RF})$
1	0	0	10	10
2			14.5	
3				
4				
5				
6				

$\gamma = 0.9$

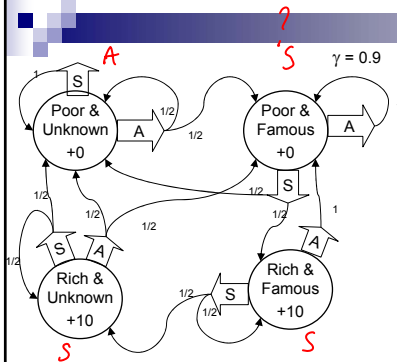
$$V_2(\text{RU}) = \max_{\mathbf{a}} R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' | \mathbf{x}, \mathbf{a}) V_1(\mathbf{x}')$$

$\mathbf{a} = \text{A} = 10 + \gamma (0.5 V_1(\text{PU}) + 0.5 V_1(\text{PF})) = 10$   
 $\mathbf{a} = \text{S} = 10 + \gamma (0.5 V_1(\text{RU}) + 0.5 V_1(\text{PU})) = 10 + 8 = 14.5$

©2005-2007 Carlos Guestrin

13

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



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

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

©2005-2007 Carlos Guestrin

14

# 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, (other possibilities: 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 – 10701/15781

Carlos Guestrin

Carnegie Mellon University

November 29<sup>th</sup>, 2007

17

## The Reinforcement Learning task

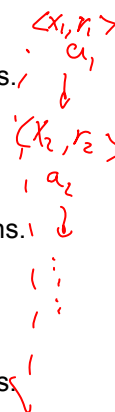
**World:** You are in state 34.  
Your immediate reward is 3. You have possible 3 actions.

**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

*time 1* ( $x=27, r=-3, a=2$ )  
*2* ( $x=33, r=7, a=5$ )  
*3* ( $x=84, r=-1000, a=8$ )  
*4* ( $x=5, r=10, a=2$ )  
 could learn  
 $P(x'|x,a), R(x,a)$   
 then use Value Iteration

# 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
" " " <u>26,</u>	" = <u>100,</u>	

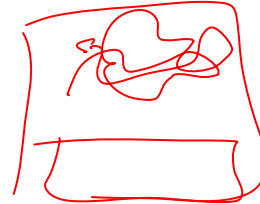


Yippee! I got to a state with a big reward! But which of my actions along the way actually helped me get there??

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

23

## Given a dataset – learn model

Given data, learn (MDP) Representation:

- Dataset:  $\langle x_1, r_1, a_1, x_2 \rangle$   
 $\langle x_2, r_2, a_2, x_3 \rangle$   
 $\vdots$
- Learn reward function:  
  $R(x,a)$   $\Leftrightarrow R: X, A \rightarrow \mathbb{R}$
- Learn transition model:  
  $P(x'|x,a)$   $= \frac{\text{count}(x'=1, x=2, a=3)}{\text{count}(x=2, a=3)}$



## 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 → 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 ( $R_{\min}$ )?
    - plug in largest reward ( $R_{\max}$ )?
  - don't know a particular transition probability?

$$P(x'|x,a)$$

## 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

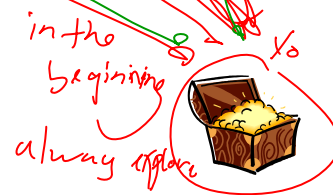
# A surprisingly simple approach for model based RL – The Rmax algorithm [Brafman & Tenenholz]

## ■ 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  $R_{max}$ !!!  $R(x,a) = R_{max}$
- 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  $x_0$ !!!
  - $R(x_0,a) = R_{max}$
  - $P(x_0|x_0,a) = 1$

# 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  $x_0$  to MDP
  - $R(x,a) = R_{\max}, \forall x,a$
  - $P(x_0|x,a) = 1, \forall x,a$
  - all states (except for  $x_0$ ) are **unknown**
- Repeat *optimal*
  - 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 $P(\mathbf{x}'|\mathbf{x},\mathbf{a})$ ?

- How many times are enough?
  - use Chernoff Bound!
- **Chernoff Bound:**
  - $X_1, \dots, X_n$  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  $1-\delta$ ,  $R_{\max}$  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 → num. states and actions is finite, so can't take too long before all states are known

*can only happen a poly number of times*

# Announcements

## ■ University Course Assessments

- Please, please, please, please, please, please, please, please, please, please, please, please, please, please...

## ■ Project:

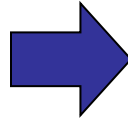
- Poster session: Tomorrow 2-4:45pm, NSH Atrium
  - please arrive a 15mins early to set up
- Paper: Friday December 14<sup>th</sup> by 2pm
  - electronic submission by email to instructors list
  - maximum of 8 pages, NIPS format
  - no late days allowed

*no late minutes*

TD-Learning and  
Q-learning – Model-free  
approaches

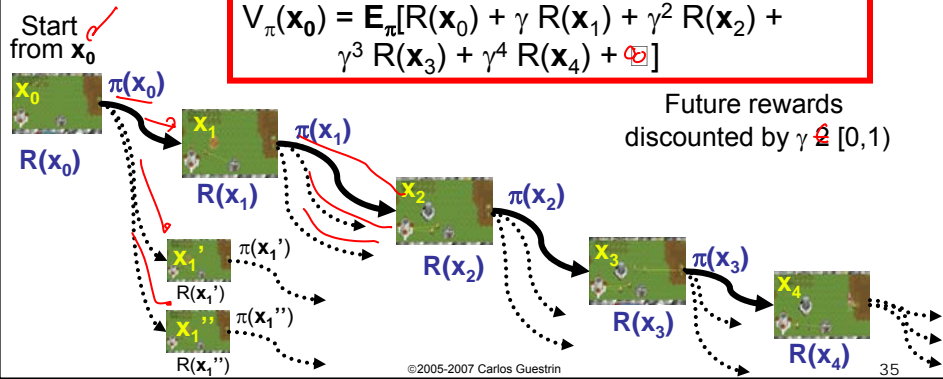
# Value of Policy

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



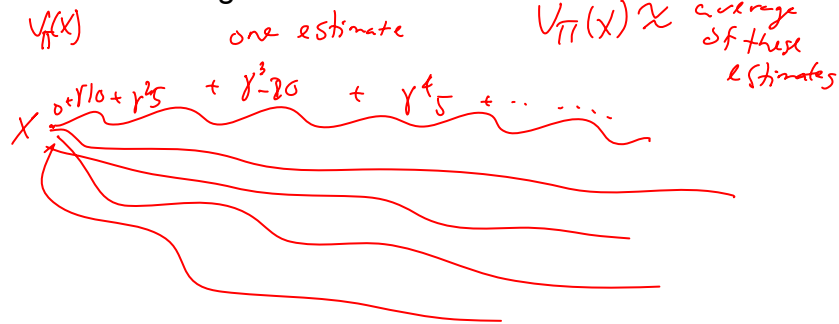
Expected long-term reward starting from  $\mathbf{x}$

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



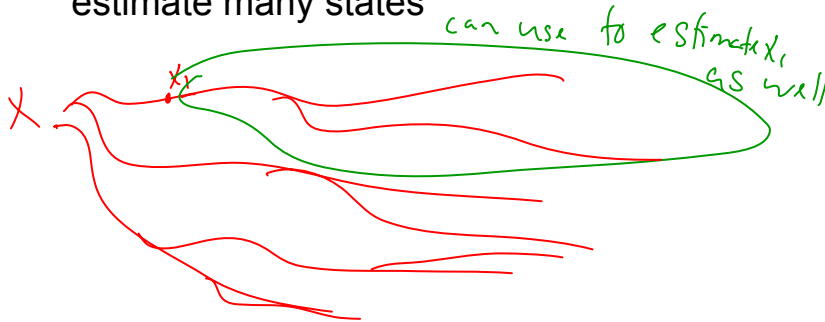
# A simple monte-carlo policy evaluation

- Estimate  $V_{\pi}(\mathbf{x})$ , start several trajectories from  $\mathbf{x}$  !  
 $V_{\pi}(\mathbf{x})$  is average reward from these trajectories
  - Hoeffding's inequality tells you how many you need
  - discounted reward ~~is~~ 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:

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

- Expressed as an expectation over next states:

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

- Initialize value function (zeros, at random, .. *Expected value for next state*)
- Idea 1: Observe a transition:  $x_t, x_{t+1}, r_{t+1}$ , approximate expec. with single sample:

$$V(x_t) = r_{t+1} + \gamma V(x_{t+1})$$

- unbiased!!
- but a very bad estimate!!!

$V_{\pi}(x_{t+1})$  is an unbiased estimate of  $E[V_{\pi}(x') | x_t]$

## Simple fix: Temporal Difference (TD) Learning [Sutton '84]

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

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

$$V_{\pi}(x_t) = (1-\alpha) \cdot V_{\pi}(x_t) + \alpha [r_{t+1} + \gamma V_{\pi}(x_{t+1})]$$

- $\alpha > 0$  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:

- $\sum_{i=1}^{\infty} \alpha_i = \infty$
- $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$

## Another model-free RL approach: Q-learning [Watkins & Dayan '92]

- TD is just for one policy...
  - How do we find the optimal policy?
  
- Q-learning:
  - Simple modification to TD
  - Learns optimal value function (and policy), not just value of fixed policy
  - Solution (almost) independent of policy you execute!

## Recall Value Iteration

■ Value iteration:  $V_{t+1}(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' | \mathbf{x}, \mathbf{a}) V_t(\mathbf{x}')$

■ Or:  $Q_{t+1}(\mathbf{x}, \mathbf{a}) = R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' | \mathbf{x}, \mathbf{a}) V_t(\mathbf{x}')$

$V_{t+1}(\mathbf{x}) = \max_{\mathbf{a}} Q_{t+1}(\mathbf{x}, \mathbf{a})$

if I know  $Q^*(x,a)$   
 $\pi^*(x) = \operatorname{argmax}_a Q^*(x,a)$

- Writing in terms of Q-function:

$Q_{t+1}(\mathbf{x}, \mathbf{a}) = R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' | \mathbf{x}, \mathbf{a}) \max_{\mathbf{a}'} Q_t(\mathbf{x}', \mathbf{a}')$

# Q-learning

$$Q_{t+1}(\mathbf{x}, \mathbf{a}) = R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' | \mathbf{x}, \mathbf{a}) \max_{\mathbf{a}'} Q_t(\mathbf{x}', \mathbf{a}')$$

*initialize  $Q_0(x_a)$  to eq. 0*

- Observe a transition:  $\mathbf{x}_t, \mathbf{a}_t, \mathbf{x}_{t+1}, r_{t+1}$ , approximate expectation by mixture of new sample with old estimate:

- transition now from state-action pair to next state and reward

$$Q(x_t, a_t) = (1-\alpha) Q(x_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(x_{t+1}, a)]$$

*$\approx V(x_{t+1})$*

- $\alpha > 0$  is learning rate

# Q-learning convergence

- Under same conditions as TD, Q-learning converges to optimal value function  $Q^*$

- Can run any policy, as long as policy visits every state-action pair infinitely often
- Typical policies (non of these address Exploration-Exploitation tradeoff)

- $\epsilon$ -greedy:  $\mathbf{a}_t = \arg \max_{\mathbf{a}} Q_t(\mathbf{x}, \mathbf{a})$

- with prob.  $(1-\epsilon)$  take greedy action:

- with prob.  $\epsilon$  take an action at (uniformly) random

*t ← time step*

- Boltzmann (softmax) policy:

$$P(\mathbf{a}_t | \mathbf{x}) \propto \exp\left\{ \frac{Q_t(\mathbf{x}, \mathbf{a})}{K} \right\}$$

- K – “temperature” parameter, ~~K=0, as !!!~~  
*K → ∞ as t → ∞*

# The curse of dimensionality:

## A significant challenge in MDPs and RL

- MDPs and RL are polynomial in number of states and actions

- Consider a game with n units (e.g., peasants, footmen, etc.)

- How many states?  $k^n$
- How many actions?  $|A|^n$

*in k position*

*(A) actions*

- **Complexity is exponential in the number of variables used to define state!!!**

# Addressing the curse!

- Some solutions for the curse of dimensionality:

- Learning the value function: mapping from state-action pairs to values (real numbers)

- **A regression problem!**, linear R., DT, NN, NNets, ...

- Learning a policy: mapping from states to actions

- **A classification problem!**

$\Pi: X_s \rightarrow A$

- Use many of the ideas you learned this semester:

- linear regression, SVMs, decision trees, neural networks, Bayes nets, etc.!!!

*For example: TD Games; TD Learning + Neural Net representation for V*

## What you need to know about RL

### ■ A model-based approach:

- address exploration-exploitation tradeoff and credit assignment problem
- the R-max algorithm

### ■ A model-free approach:

- never needs to learn transition model and reward function
- TD-learning
- Q-learning

# Closing....

Machine Learning – 10701/15781

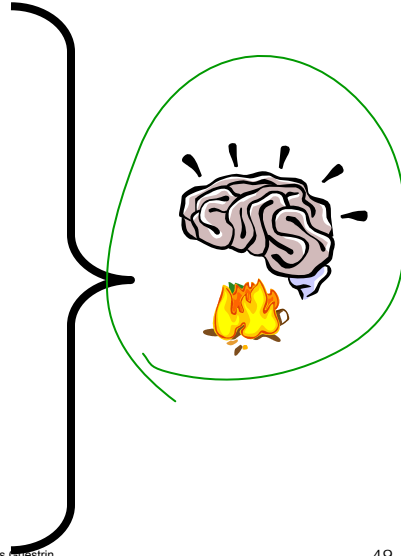
Carlos Guestrin

Carnegie Mellon University

November 29<sup>th</sup>, 2007

# What you have learned this semester

- Learning is function approximation
- Point estimation
- Regression
- Discriminative v. Generative learning
- Naïve Bayes
- Logistic regression
- Bias-Variance tradeoff
- Neural nets
- Decision trees
- Cross validation
- Boosting
- Instance-based learning
- SVMs
- Kernel trick
- PAC learning
- VC dimension
- Mistake bounds
- Bayes nets
  - representation, inference, parameter and structure learning
- HMMs
  - representation, inference, learning
- K-means
- EM
- Feature selection, dimensionality reduction, PCA
- MDPs
- Reinforcement learning



# BIG PICTURE

- Improving the performance at some task though experience!!! 😊
  - before you start any learning task, remember the fundamental questions:

**What is the learning problem?**

**From what experience?**

**What model?**

**What loss function are you optimizing?**

**With what optimization algorithm?**

**Which learning algorithm?**

**With what guarantees?**

**How will you evaluate it?**

# What next?

- Machine Learning Lunch talks: <http://www.cs.cmu.edu/~learning/>
- Intelligence Seminars: <http://www.cs.cmu.edu/~iseminar/>
- Journal:
  - JMLR – Journal of Machine Learning Research (free, on the web)
- Conferences:
  - ICML: International Conference on Machine Learning
  - NIPS: Neural Information Processing Systems
  - COLT: Computational Learning Theory
  - UAI: Uncertainty in AI
  - AIStats: intersection of Statistics and AI
  - Also AAAI, IJCAI and others
- Some MLD courses:
  - 10-708 Probabilistic Graphical Models (Fall) *Spring 2009*
  - 10-705 Intermediate Statistics (Fall)
  - 11-762 Language and Statistics II (Fall)
  - 10-702 Statistical Foundations of Machine Learning (Spring)
  - 10-705 Optimization (Spring)
  - ...

# You have done a lot!!!

- And (hopefully) learned a lot!!!
  - Implemented
    - NB
    - LR
    - Nearest Neighbors
    - Boosting
    - SVM
    - HMMs
    - PCA
    - EM and GMM
  - Answered hard questions and proved many interesting results
  - Completed (I am sure) an amazing ML project

**Thank You for the  
Hard Work!!!**