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

Markov Decision Processes (MDPs)

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

May 1st, 2006

Announcements

- Project:
  - Poster session: Friday May 5th 2-5pm, NSH Atrium
    - please arrive a little early to set up
- FCEs!!!!
  - Please, please, please, please, please, please give us your feedback, it helps us improve the class! ☺
    - 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 γ" is $\gamma \in [0, 1)$

(reward now) +

γ (reward in 1 time step) +

γ^2 (reward in 2 time steps) +

γ^3 (reward in 3 time steps) +

:

: (infinite sum)

for example:

$$20 + \gamma.20 + \gamma^2.20 + \gamma^3.20 + \vdots$$

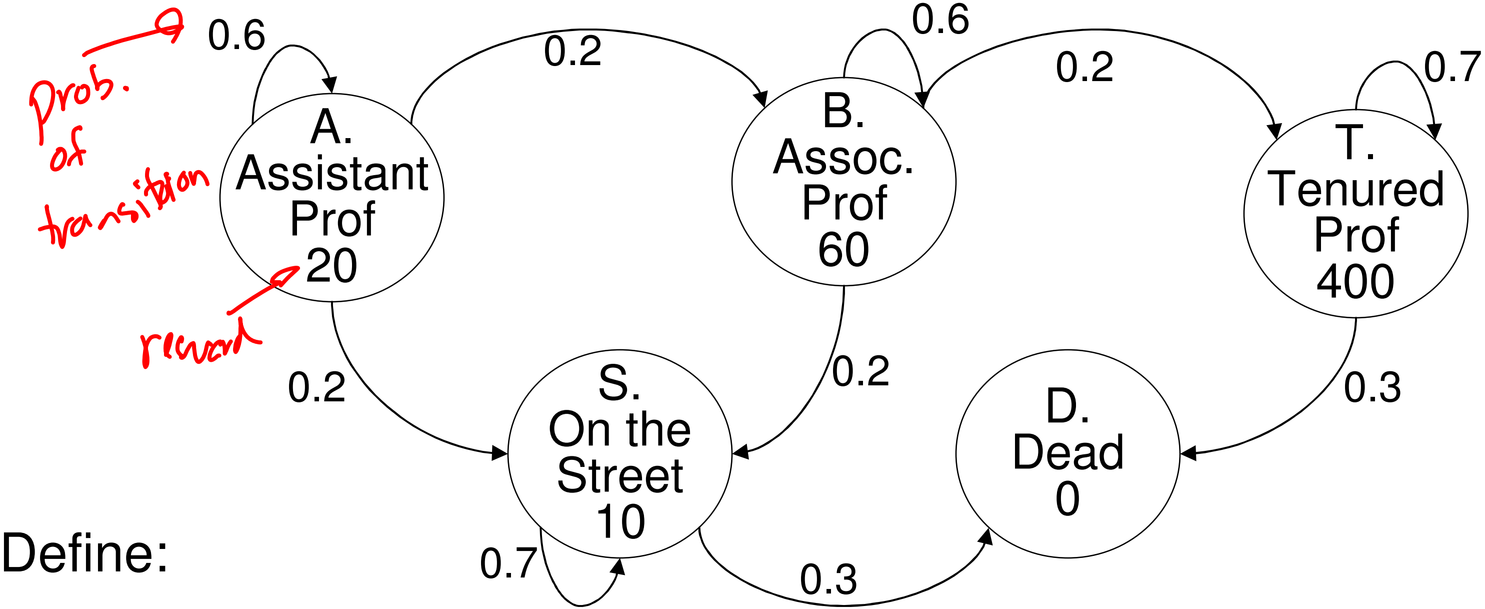
geometric series

$$= \frac{20}{1-\gamma} = \frac{20}{1-0.9} = 200$$

The Academic Life

Simple Markov Chain

Assume Discount Factor γ = 0.9



Define:

V_A = Expected discounted future rewards starting in state A

V_B = Expected discounted future rewards starting in state B

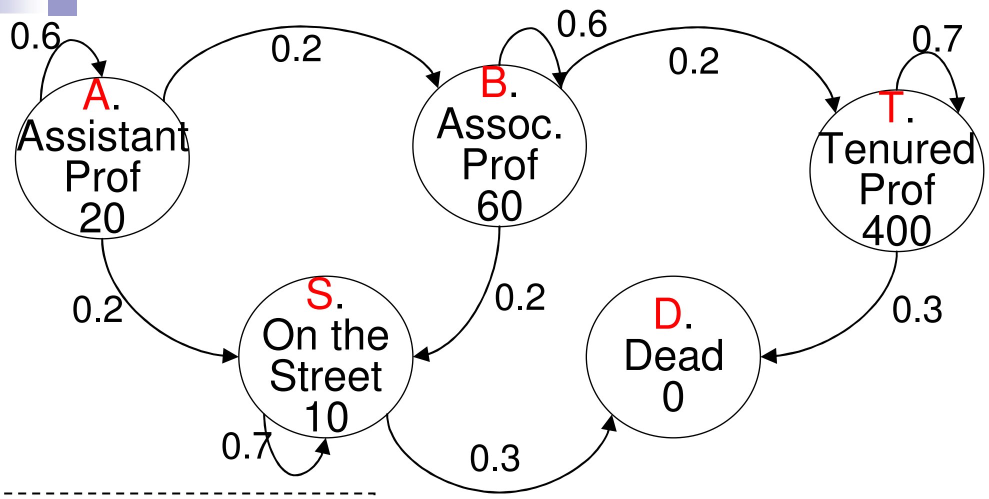
V_T = " " " " " " " T

V_S = " " " " " " " S

V_D = " " " " " " " D

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

Computing the Future Rewards of an Academic



Assume Discount Factor γ = 0.9

$V_D = 0$

V_T T D $V_D = 0$

$$V_T = \underset{\text{first year}}{400} + \underset{\text{second year}}{\gamma}\left[0.3 \cdot \underset{0}{V_D} + 0.7 V_T\right]$$

$$\Rightarrow V_T = \frac{400}{1 - 0.7\gamma}$$

$$V_B = 60 + \gamma\left[0.6 V_B + 0.2 V_T + 0.2 V_S\right]$$

$$V_S = 10 + \gamma\left[0.7 V_S + 0.3 \underset{0}{V_D}\right]$$

Joint Decision Space

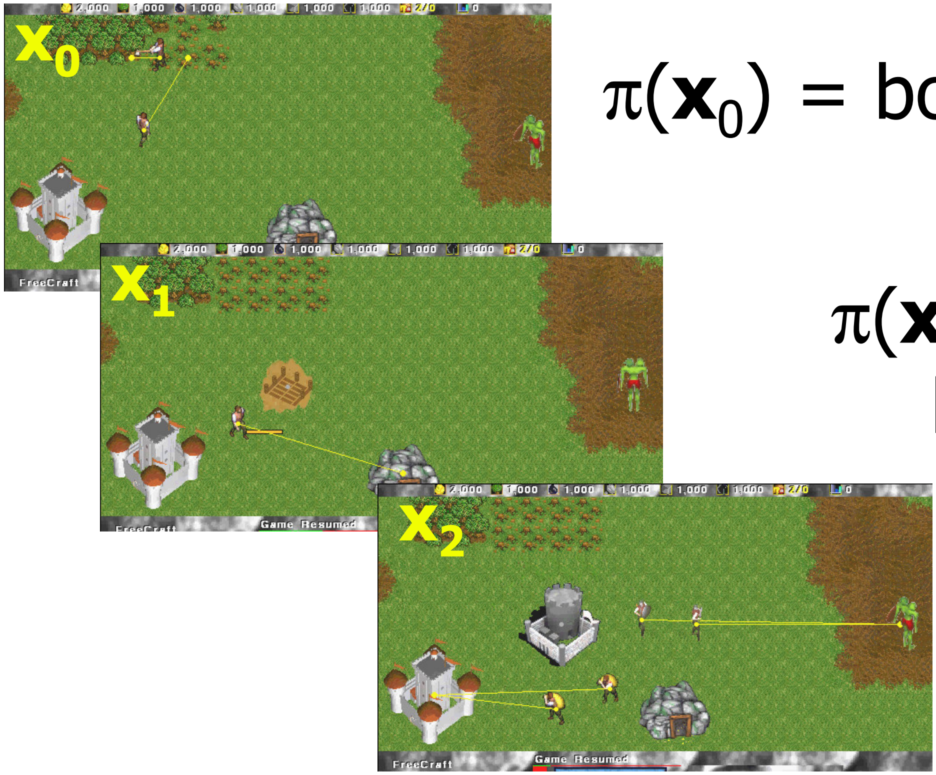
Markov Decision Process (MDP) Representation:

- 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**’|**x**,**a**)

Policy

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

At state **x**, action **a** for all agents



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

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

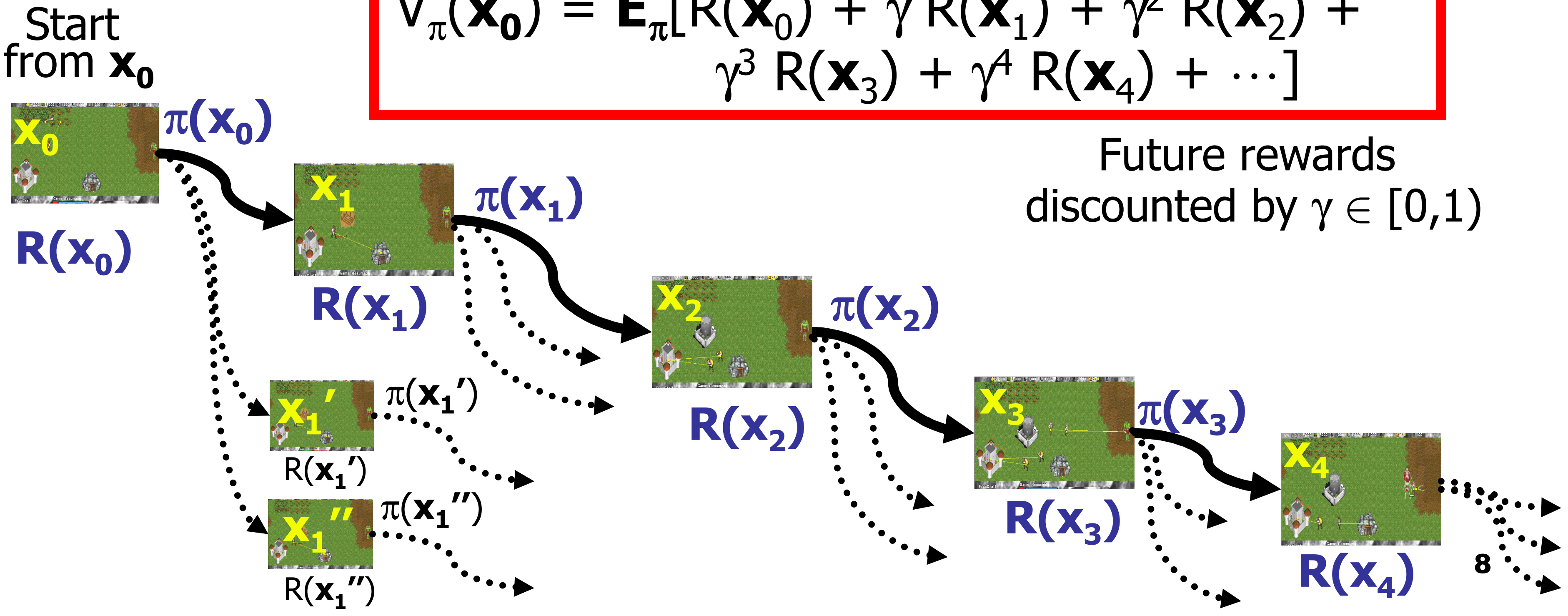
$\pi(\mathbf{x}_2)$ = peasants get gold, footmen attack

Value of Policy

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

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

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



Future rewards discounted by $\gamma \in [0,1)$

Computing the value of a policy

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

- 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!

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:

Computing the value of a policy 2 – iteratively

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

- If you have 1000,000 states, inverting a 1000,000x1000,000 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 $||V_\pi-V_{t+1}||_\infty \leq \varepsilon/(1-\gamma)$

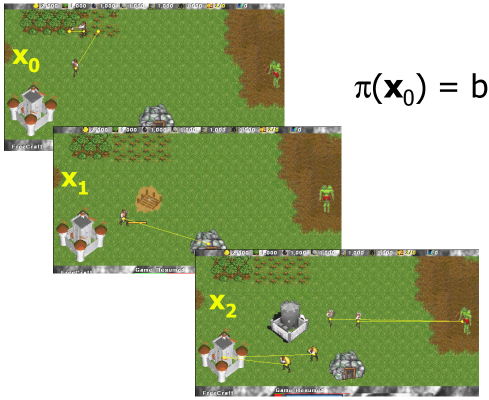
But we want to learn a **Policy**

- 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 **x**, action **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

Another recursion!

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

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

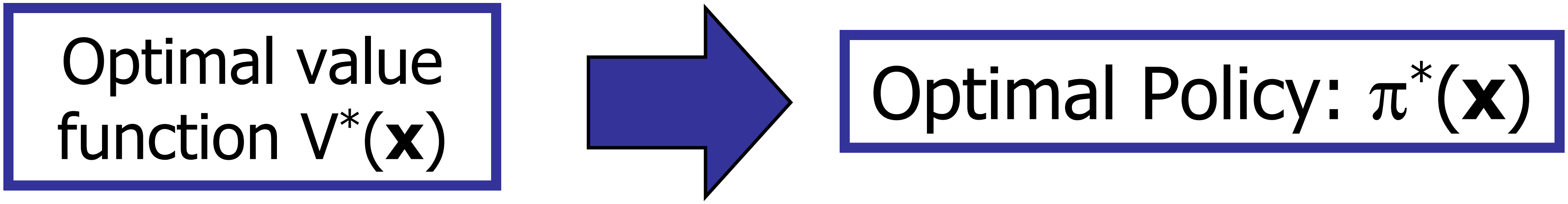
- Evaluating policy π:

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

- Computing the optimal value V* - Bellman equation

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

Optimal Long-term Plan



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

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

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

Optimal policy:

$$\pi^{*}(\mathbf{x}) = \arg\max_{\mathbf{a}} Q^{*}(\mathbf{x},\mathbf{a})$$

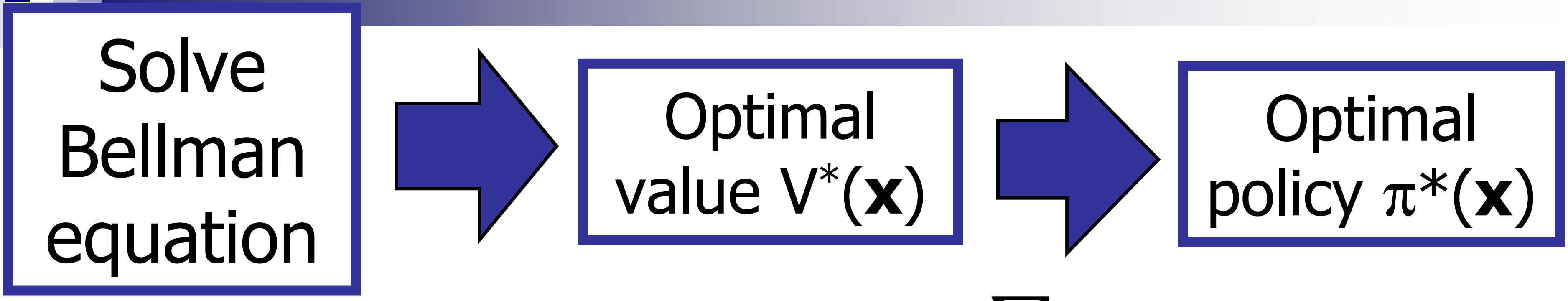
Interesting fact – Unique value

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

- *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

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

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

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

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

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

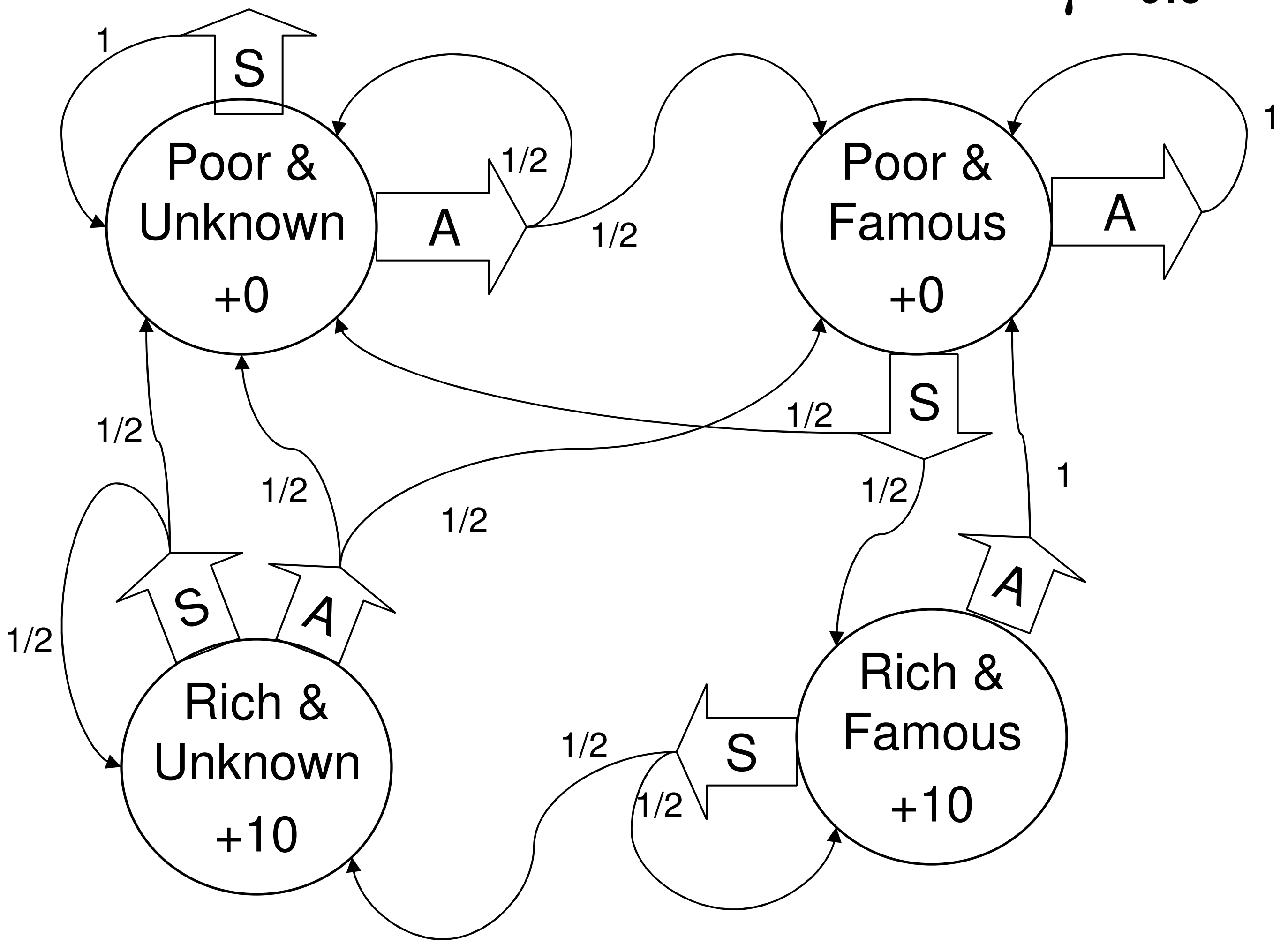
- Start with some guess V_0
- Iteratively say:
  - $V_{t+1}(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' | \mathbf{x},\mathbf{a}) V_t(\mathbf{x}')$
- Stop when $||V_{t+1}-V_t||_\infty \leq \varepsilon$
  - means that $||V^*-V_{t+1}||_\infty \leq \varepsilon/(1-\gamma)$

A simple example

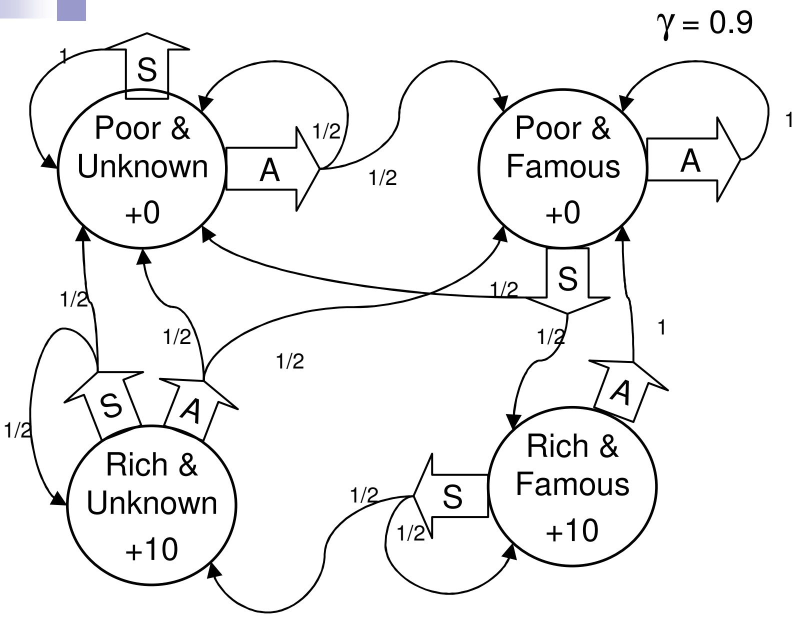
You run a startup company.

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

$\gamma = 0.9$



Let's compute V_t(x) for our example

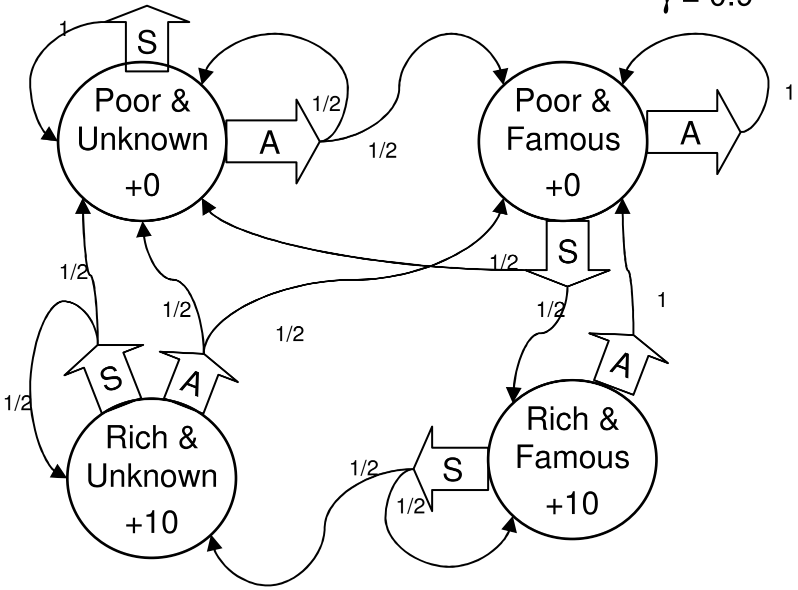


t	V_t(PU)	V_t(PF)	V_t(RU)	V_t(RF)
1				
2				
3				
4				
5				
6				

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

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

$\gamma = 0.9$



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	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

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

Policy iteration – Another approach for computing π^*

- Start with some guess for a policy π_0
- Iteratively say:
  - evaluate policy: $V_t(\mathbf{x}) = R(\mathbf{x}, \mathbf{a} = \pi_t(\mathbf{x})) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a} = \pi_t(\mathbf{x})) V_t(\mathbf{x}')$
  - improve policy: $\pi_{t+1}(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) V_t(\mathbf{x}')$
- Stop when
  - □ policy stops changing
    - usually happens in about 10 iterations
  - □ or $||V_{t+1}-V_t||_\infty \leq \varepsilon$
    - means that $||V^*-V_{t+1}||_\infty \leq \varepsilon/(1-\gamma)$

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

3rd Approach

Linear Programming

LP Solution to MDP

[Manne ‘60]

Value computed by linear programming:

$$\text{minimize}: \sum_{\mathbf{x}} V(\mathbf{x})$$

$$\text{subject to}: \begin{cases} V(\mathbf{x}) \geq \; R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V(\mathbf{x}') \\ \forall \mathbf{x},\mathbf{a} \end{cases}$$

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

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

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

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.

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 → 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

Brafman & Tennenholtz 2002
(see class website)

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**)

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?

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

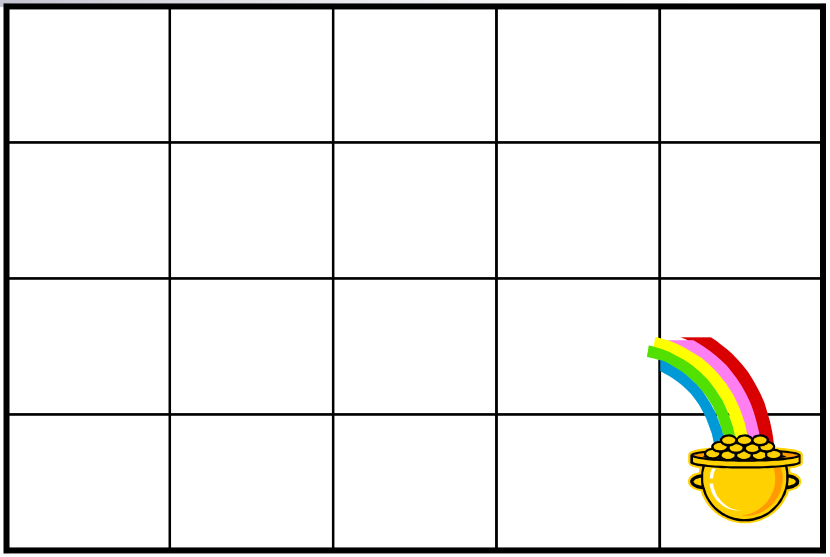
- A strong advantage of a model-based approach:
  - you know which states estimate for rewards and transitions are bad
  - can (try) to plan to reach these states
  - have a good estimate of how long it takes to get there

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 **x**,**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

Problems with model-based approach

- If state space is large
  - transition matrix is very large!
  - requires many visits to declare a state as know
- Hard to do “approximate” learning with large state spaces
  - some options exist, though

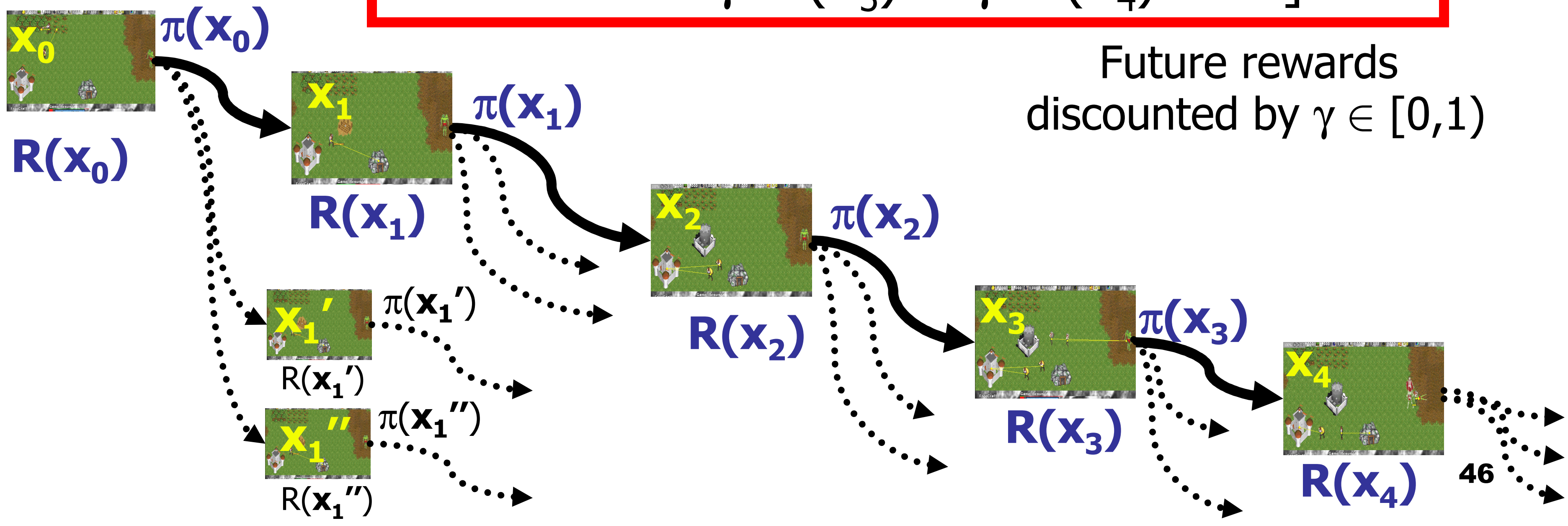
TD-Learning and Q-learning – Model-free approaches

Value of Policy

Value: $V_\pi(\mathbf{x})$ → Expected long-term reward starting from **x**

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

Start from $\mathbf{x_0}$



Future rewards discounted by $\gamma \in [0,1)$

A simple monte-carlo policy evaluation

- Estimate V(**x**), start several trajectories from **x** → V(**x**) is average reward from these trajectories
  - □ Hoeffding's inequality tells you how many you need
  - □ discounted reward → 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:

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

- Expressed as an expectation over next states:

$$V_\pi(x) \quad = \quad R(x) + \gamma E\left[V_\pi(x') \mid x, a = \pi(x) \right]$$

- Initialize value function (zeros, at random,…)
- Idea 1: Observe a transition: $\mathbf{x}_t \rightarrow \mathbf{x}_{t+1}, \mathbf{r}_{t+1}$, approximate expec. with single sample:
  - □ unbiased!!
  - □ but a very bad estimate!!!

Simple fix: Temporal Difference (TD) Learning

- Idea 2: Observe a transition: $\mathbf{x}_t \rightarrow \mathbf{x}_{t+1}, \mathbf{r}_{t+1}$, approximate expec. by mixture of new sample with old estimate:

  - □ $\alpha>0$ is learning rate

TD converges (can take a long time!!!)

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

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